April 2009

digit Fast Track to AJAX

Introduction

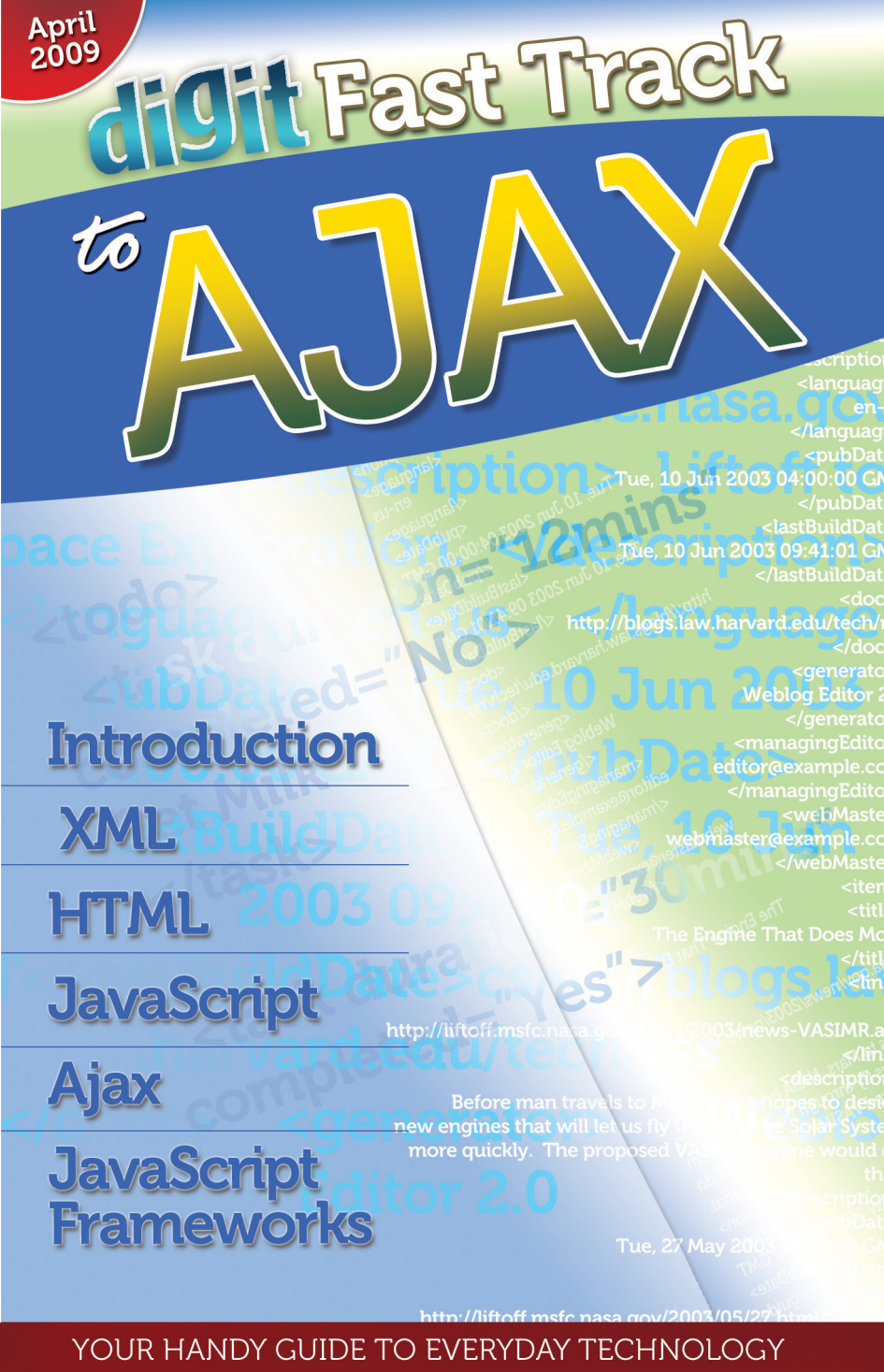
XML

HTML

JavaScript

Ajax

JavaScript Frameworks



YOUR HANDY GUIDE TO EVERYDAY TECHNOLOGY

Fast Track to

AJAX

By Team Digit

Credits

The People Behind This Book

EDITORIAL

Editor-in-chief	**Edward Henning**
Assistant Editor	**Robert Sovereign-Smith**
Editor-at-Large and Online Architect	**Ahmed Shaikh**
Head-Copy Desk	**Nash David**
Writer	**Kshitij Sobti**

DESIGN AND LAYOUT

Layout Design	**MV Sajeev**
Cover Design	**Rohit A Chandwaskar**

Published by 9.9 Interactive

April 2009

Free with Digit. Not to be sold separately. If you have paid separately for this book, please email the editor at **editor@thinkdigit.com** along with details of location of purchase, for appropriate action.

CONTENTS

Introduction to AJAX

1.1 History of AJAX

Ajax is a concept — a way of making the web more interactive, and as such was never really created or introduced. In fact, the concept existed much before it became popular. However, the term was coined in 2005, by Jesse James Garrett, founder of Adaptive Path.

With the introduction of the iframe element in HTML by Internet Explorer in 1996, web developers can now asynchronously load another web page in a section of a currently loaded page. By using an iframe with zero dimensions, you could use an iframe to load content from another page, extract the relevant parts and inject it into your own page, all in the background.

In 1999, the new XMLHttpRequest ActiveX control was developed by Microsoft for Internet Explorer. The new object can asynchronously load pages, from any web site as desired by the developer. As of now, this is available in all major browsers such as Firefox, Safari and Opera.

In 2006, this control was standardised by the world wide web consortium (W3C), in a draft specification. As a result, once finalised, any new browser that plans to follow web standards will have to support this object.

1.2 Setting it up

JavaScript, XML and HTML are pure text formats, and the only programming tool you really need is a text editor. If you prefer manual coding, you should either prefer Notepad++, or a pure text editor.

An integrated development environment (IDE) is unnecessary in this case, as you will not even need to compile anything. However, working with one improves productivity as it will support syntax highlighting, code completion, and will warn you if you mistype anything.

Unless you plan to do some really heavy design work for your web site, Aptana Studio is a good choice. It isn't a WYSIWYG editor, which may be a turn-off if you intend to make complicated layouts. However, for getting your layout off the ground, you can use free and open source applications such as Komposer, or Amaya. If you you really want to go for web development, Dreamweaver is a professional solution by Adobe.

So to effectively use this Fast Track to AJAX, you should have the following software installed on your computer:

- Aptana Studio
- Notepad++
- A Browser (preferably Mozilla Firefox with the FireBug addon)

XML

2.1 Introduction

XML is all over the place these days, whether you're checking out an RSS or Atom field, or using Microsoft Office 2007's new DOCX format. The reason for its popularity stems from the fact that it is an open standard that's highly extensible and has a wide range of tools and techniques available for processing it.

One thing you should know about XML is — by itself, it doesn't define a language. Instead, it's a specification of how one may create a language. It's very flexible, and can be used to represent almost any kind of data, as such it is a perfect candidate for data transfer over the internet. As long as the data follows the XML specification, the creator can be sure that any tool capable of processing XML will be able to decode it.

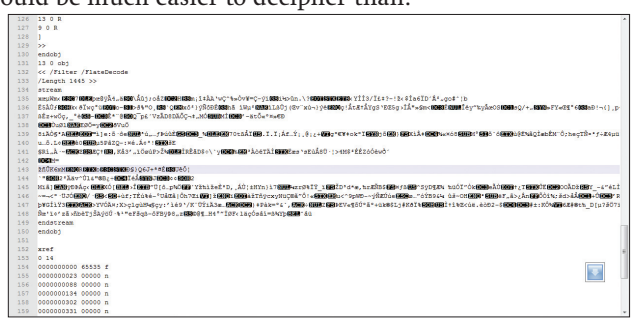
Over time, many data formats have been lost as developers did not document how data structures are stored within a file. XML is plain text and hence human readable. Therefore, such a problem is eliminated, and data in most cases documents itself. Let's look at a common XML format called RSS.

```
<?xml version="1.0"?>
<rss version="2.0">
  <channel>
    <title>
      Liftoff News
    </title>
    <link>
      http://liftoff.msfc.nasa.gov/
    </link>
    <description>
      Liftoff to Space Exploration.
    </description>
```

```
        <language>
          en-us
        </language>
        <pubDate>
          Tue, 10 Jun 2003 04:00:00 GMT
        </pubDate>
        <lastBuildDate>
          Tue, 10 Jun 2003 09:41:01 GMT
        </lastBuildDate>
        <docs>
          http://blogs.law.harvard.edu/tech/rss
        </docs>
        <generator>
          Weblog Editor 2.0
        </generator>
        <managingEditor>
          editor@example.com
        </managingEditor>
        <webMaster>
          webmaster@example.com
        </webMaster>
        <item>
          <title>
            The Engine That Does More
          </title>
          <link>
              http://liftoff.msfc.nasa.gov/
news/2003/news-VASIMR.asp
          </link>
          <description>
              Before man travels to Mars, NASA
hopes to design new engines that will let us
fly through the Solar System more quickly.  The
proposed VASIMR engine would do that.
          </description>
          <pubDate>
            Tue, 27 May 2003 08:37:32 GMT
          </pubDate>
          <guid>
        http://liftoff.msfc.nasa.gov/2003/05/27.
```

```
html#item571
          </guid>
        </item>
        <item>
          <title>
            Astronauts' Dirty Laundry
          </title>
          <link>
              http://liftoff.msfc.nasa.gov/
news/2003/news-laundry.asp
          </link>
          <description>
            Compared to earlier spacecraft, the
International Space Station has many luxuries,
but laundry facilities are not one of them.
Instead, astronauts have other options.
          </description>
          <pubDate>
            Tue, 20 May 2003 08:56:02 GMT
          </pubDate>
          <guid>
        http://liftoff.msfc.nasa.gov/2003/05/20.
html#item570
          </guid>
        </item>
      </channel>
```

At first glance, this may seem complicated. However, in the future, if all specifications of XML and RSS are lost, this would be much easier to decipher than:



This is a screenshot of a PDF file opened in Notepad++. Perhaps only a seasoned developer who has worked a lot on PDF might be able to recognise it! You can take any random section of XML data and get some information out of it.

So let's start with the basic rules of XML. Here, we will focus only on the basic features required to make a valid XML file suitable for use in AJAX.

2.2 The XML structure

XML is a tag-based language — the data that you need to represent has to be grouped and organised within tags. It is very easy to represent structured data in XML. Let's consider an example of data we use every day, and see how it can possibly to represented in XML.

2.2.1 A simple to-do list

```
<todo>
  <task duration="12mins" completed="No">
 Get Milk
  </task>

  <task duration="30mins" completed="Yes">
 Dry-clean Rug
  </task>

  <task duration="5hrs" completed="No">
 Complete report
  </task>
</todo>
```

2.2.2Contact data

```
<addressbook>
  <contact>
    <name>
      John Doe
    </name>
    <phone type="home">
      01122245656
    </phone>
```

```
        <phone type="mobile">
          +919912345567
        </phone>
        <email>
          john.doe@gmail.com
        </email>
        <email>
          john_doe_43@ymail.com
        </email>
        <address>
    123,St. 4,Plot 5,Sixganj,Sevenpur,MP,462006
        </address>
        <image>
          /contacts-images/jdoe.png
        </image>
      </contact>
      .
      .
      .
   </addressbook>
```

2.2.3 Image gallery:

This following is a trimmed down version of how Picasa exports galleries in the XML format.

```
   <album>
     <albumName>
       School Trip
     </albumName>
     <albumItemCount>
       3
     </albumItemCount>
     <albumCaption>
         Pictures taken during school trip to
Dehradun.
     </albumCaption>
     <images>
       <image>
         <itemLargeImage>
           images/image315.jpg
         </itemLargeImage>
```

```
          <itemWidth>
            240
          </itemWidth>
          <itemHeight>
            320
          </itemHeight>
          <itemName>
            image315.jpg
          </itemName>
          <itemNumber>
            36
          </itemNumber>
          <itemOriginalPath>
             C:\Users\User666\Pictures\2008\july
12\image315.jpg
          </itemOriginalPath>
          <itemCaption>
            At the train station.
          </itemCaption>
          <itemSize>
            245kb
          </itemSize>
        </image>
        .
        .
        .
    </images>
  </album>
```

2.3 The XML syntax

If you decide to include a new feature in an app, say for example, enabling your contacts to now store their images. All you would need to do is add a tag for it and you're done! This new contact data file will continue to work with the older app as the older version will just ignore the tag for the contact image. Or if you decide that your to-do app should now have sub-tasks, you could have a to-do tag inside another to-do tag, and depending on how your previous application is designed, it would either ignore the sub-tasks or show all of

them unstructured.

Let us go over some rules for complying with the XML standard.

- An XML document can have only one root tag. This is shown as follows.

Correct Syntax	Incorrect Syntax
<todo> <task duration="12mins" completed="No"> Get Milk </task> <task duration="30mins" completed="Yes"> Dry-clean Rug </task> <task duration="5hrs" completed="No"> Complete report </task> </todo>	<task duration="12mins" completed="No"> Get Milk </task> <task duration="30mins" completed="Yes"> Dry-clean Rug </task> <task duration="5hrs" completed="No"> Complete report </task>

- Each opening element must have a closing element.

Correct Syntax	Incorrect Syntax
<todo> <task duration="12mins" completed="No"> Get Milk </task> <task duration="30mins" completed="Yes"> Dry-clean Rug </task> <task duration="5hrs" completed="No"> Complete report </task> </todo>	<todo> <taskduration="12mins" completed="No"> Get Milk <taskduration="30mins" completed="Yes"> Dry-clean Rug </task> <task duration="5hrs" completed="No"> Complete report </task>

- Element attributes must be enclosed in single or double quotes.

Correct Syntax	Incorrect Syntax
<todo> <task duration="12mins" completed="No"> Get Milk </task> <task duration="30mins" completed='Yes'> Dry-clean Rug </task> <task duration="5hrs" completed='No'> Complete report </task> </todo>	<todo> <task duration='12mins' completed="No"> Get Milk </task> <taskduration="30mins" completed="Yes"> Dry-clean Rug </task> <task duration="5hrs" completed="No"> Complete report </task> </todo>

- Each attribute may occur only one time.

Correct Syntax	Incorrect Syntax
<todo> <task duration="12mins" completed="No"> Get Milk </task> <task duration="30mins" completed="Yes"> Dry-clean Rug </task> <task duration="5hrs" completed= "No"> Complete report </task> </todo>	<todo> <taskduration="12mins" completed="No"> Get Milk </task> <taskduration="30mins" completed='Yes'> Dry-clean Rug </task> <task duration="5hrs" duration="15hrs" completed='No'> Complete report </task> </todo>

- Tags must be properly nested.

Correct Syntax	Incorrect Syntax
<todo> <task duration="12mins" completed="No"> Get Milk </task> <task duration="30mins" completed='Yes'> Dry-clean Rug </task> <task duration="5hrs" completed='No'> Complete report </task> </todo>	<todo> <task duration="5hrs" completed='No'> Complete report </todo> </task>

- Element names are case-sensitive

Correct Syntax	Incorrect Syntax
<todo> <task duration="12mins" completed="No"> Get Milk </task> <task duration="30mins" completed='Yes'> Dry-clean Rug </task> <task duration="5hrs" completed='No'> Complete report </task> </todo>	<todo> <taskduration="12mins" completed="No"> Get Milk </TASK> <tasKduration="30mins" completed='Yes'> Dry-clean Rug </task> <task duration="5hrs" completed='No'> Complete report </task> </todo>

These are the rules for a document to conform to the XML standard. However, depending on the nature or use of the XML document, further restrictions may apply. For example, in case of RSS feeds, the tag names, and the way they are supposed to be nested is clearly defined.

XHTML is another example of an XML-compliant format with extra requirements. XHTML is not a new language; it is merely an update to HTML, to make it compliant with the XML standard. HTML has some syntactical elements that do not abide by the XML standard. For all future purposes, HTML will follow the XML standard.

2.4 XML in AJAX

XML is a very powerful way of implementing AJAX. However, this is not essential. XML is used in AJAX primarily to retrieve data from the server, and unlike the examples shown, the XML code is usually generated on the fly using server-side scripts along with database access. RSS feeds provided by major sites are basically server-side scripts, which compile a list of items from the requested list, and puts them in an XML format that is compliant with RSS.

Server-side scripting is out of the scope of this book. However, you shouldn't feel like you have accomplished nothing. You can still work with static XML files you create, or you can use the XML feeds provided by a variety of applications found on the web.

2.4.1 Disadvantages of using XML

One of the biggest disadvantages of XML is often the fact that it has a lot of overhead. This means that the actual data content of an XML file is quite low, as a lot of space is wasted in tags. As a result, this is not a very efficient medium for transporting data.

Consider the following example:

```
<addressbook>
  <contact>
    <name>
      Tag Jones
    </name>
```

```
        <age>
          12
        </age>
        <birthday>
          11-08-1958
        </birthday>
        <address>
           34, Someplace, Some lane, Some City,
Some State, Some country
        </address>
        <email>
          abc@xyz.com
        </email>
        <private>
          1
        </private>
      </contact>
   </addressbook>
```

The additional information here is merely:

```
Tag Jones; 12; 11-08-1958; 34, Someplace,
Some lane, Some City, Some State, Some country;
abc@xyz.com; 1
```

The data is a mere 101 characters compared to the XML version, which is 247 characters — an efficiency of 41 per cent! The most inefficiently stored data is the age field, which is usually just two characters; and the private flag, which is a single character. The efficiency further reduces if the data has a lot of flags i.e. YES / NO options.

We could accomplish the same thing we have done with the above XML in a much more efficient manner by rewriting it as follows:

```
   <a>
     <c>
       <n>Tag Jones</n>
       <g>12</g>
       <b>11-08-1958</b>
       <d>
   34, Someplace, Some lane, Some City, Some
State, Some country
```

```
        </d>
        <e>abc@xyz.com</e>
        <p>1</p>
      </c>
    </a>
```

This is 159 characters, or an efficiency of approximately 64 per cent. In doing so, however, we lose the most important benefit of using XML in the first place: That is to make it easily readable by humans.

Often, when writing an AJAX application, the data is being transferred from the server to the client directly and doesn't need any human interpretation. The data is read by the client, is interpreted, and is then disposed of. As such, data being human readable isn't that important and other less verbose and more efficient formats are often adopted. The most important and popular one of them is JSON.

Another flaw that XML suffers from is that it requires significant processing before it can be used in the code. JSON, on the other hand, is JavaScript's native format of data, and as such requires little or no processing on the client side. In fact, since XML processing takes place in the background, a developer tends to use it excessively without being aware of how resource intensive it may be.

Conclusion

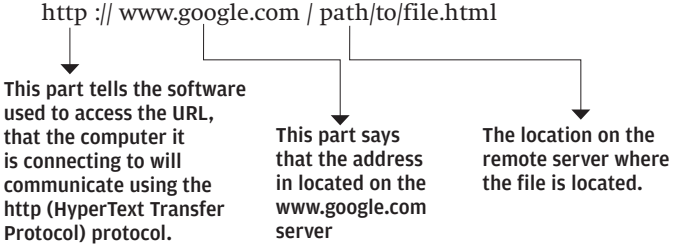
You now know enough on XML to come up with your own innovative methods for storing data. XML is flexible enough that you can keep incorporating new features without breaking old ones. However, it is also important to keep its flaws in mind.

Next, we will look at HTML, the basic platform on which you will create your AJAX application.

HTML

3.1 Introduction to the web

Before we delve deep into HTML, let us look at how the internet works. Starting with the URL, this stands for uniform resource locator. It is a uniform or standardised way of locating a resource. We have all used URLs while surfing the web, and need no further introduction. This is how a URL works:



To explain this by analogy, if you had to tell your friend to talk to Kiran Siddappaji in English and ask him about the price of one kilogram of rice, you might represent it the following way as a URL:

english://Kiran.Siddappaji/petrol/1litre/price

Thankfully, a normal conversation is a lot easier! Another important bit of information is the port number. Although it doesn't matter, sometimes it is necessary. A URL with a port is usually in the form of:

http://www.google.com:56/something/going/somewhere

56 is the port.

Usually, we need not specify a port number, as it is implied in the http part. Http is usually associated with port 80 or 81, so whether you enter **http://www.google.com:80** or **http://www.google.com** it still works the same way. However, try some other number and it may not work, unless a server is operational

on that port as well. This means that with a single web site **www.google.com** you can have multiple web sites, as ports range from `1` to `65,535`!

Finally, you are connected to a server talking in HTTP, which basically means that the pages are formatted in HTML. HTML content is now retrieved by the user, and is interpreted, processed and displayed on the users' browser.

There are other protocols besides HTTP. For example, HTTPS, UDP, news, gopher and FTP. However, since it merely tells the software application what language the destination speaks, it is usually safe to omit, as most software operate on very few.

When you open a URL in your browser, it connects to the server specified at the default or specified port, and retrieves the requested data. If it is communicating with an HTTP web site, the communication will follow a different format, as compared to an FTP server.

3.2 HTML tags

Like XML, HTML is also a tag-based language and has now been made to conform to the XML standard, with the new XHTML specification.

The HTML standard limits the tags you can use, and assigns a meaning to each tag. Some common tags are:

`html`	This is the root tag which encloses the entire body of the document.
`head`	The header area of the document, this part of the document has instructions for the browser, and is not actually displayed on the output page.
`title`	This contains the title of the page as shown in the browser window / tab.
`meta`	Contains information about the keywords and target of the data content of the page.
`script`	Contains any scripts, usually JavaScript, that are used in the page.
`noscript`	Defines content to be displayed if user browser doesn't support JavaScript.
`body`	Contains the main content of the page

	that is to be displayed in the browser window.
a	This element creates a link to the specified URL.
h1 to h6	Header 1 to 6
p	A paragraph element.
br	Line break
table	Used to represent data in a tabular form.
hr	Horizontal Rule
td	A table cell
tr	A table row, contains td tags to specify table divisions
b,u,i	Bold, underline, italicise, now depreciated in favour of CSS styling
div	Defines a section within the document
span	,,
object	Allows you to embed objects such as Flash, and Java.

3.3 Learning basic HTML

As usual, we start off with a Hello World example, just to be polite.

```
<html>
    <body>
Hello World
    </body>
</html>
```

That's it! And now that we've wished the world well, let's go on to more serious business. How about a simple list with the names of your favourite movies? And let's give it a title as well.

```
<html>
  <head>
    <title>My Favorite Movies</title>
  </head>
  <body>
    <ul>
      <li>The Matrix Series</li>
```

```
            <li>The Lord of the Rings Series</li>
            <li>Memento</li>
            <li>Star Wars Series</li>
            <li>Batman Begins</li>
            <li>The Dark Knight</li>
            <li>Gattaca</li>
            <li>Shawshank Redemption</li>
        </ul>
    </body>
</html>
```

Since this is an article on AJAX, let's see how you could use this in an AJAX scenario. You could perhaps have a textbox below the list, which allows you to add your favourite movies to the list. This example may seem a little complicated at first, but once you break it down it's really quite simple.

Let's analyse the structure of the document:

- The entire document is enclosed within an HTML tag.
- The document has a heading, which is not displayed, and other attributes such as referenced scripts and style.
- The visible part of the page is put inside the 'body' tag. This tag contains the entire visible content of the page.

In this document, we have each movie name enclosed within an `<li>` tag. The `<li>` tag is for defining list items, and as we want all of these movies to appear in a list form, we need to enclose each item inside its own `<li>` tag element. Now we put the entire list, with all the `<li>` elements inside of the `<ul>` element.

The `<ul>` tag is used to define unordered lists, i.e. bulleted lists. When you test this with a browser, you will notice that it appears as an unnumbered bulleted list. If you want the list to be numbered, all you need to do is replace the `<ul>` opening and closing tags with `<ol>`. The `<ol>` tag is for ordered lists, i.e. lists on which the number is displayed before each item. In fact, once you learn a bit of JavaScript, you could even leave the choice to the user to define how to display the list.

HTML was designed to make changes to the style without damaging the contents of the document. The title element is what decides the text on the title bar of the browser, or the tab in which the page is open. In addition, we should also include a title inside the page. We can do

this by using from the h1 to h6 tags that allow you to have multiple levels of structure within the page. For example, we could do the following:

```
<html>
  <head>
    <title>My Favorite Movies</title>
  </head>
  <body>
    <h1>A List of my favorite Movies</h1>
    <h2>Movie Series:</h2>
    <ul>
      <li>The Matrix Series</li>
      <li>The Lord of the Rings Series</li>
      <li>Star Wars Series</li>
    </ul>
    <h2>Individual Movies:</h2>
    <ul>
      <li>Memento</li>
      <li>Batman Begins</li>
      <li>The Dark Knight</li>
      <li>Gattaca</li>
      <li>Shawshank Redemption</li>
    </ul>
  </body>
</html>
```

HTML is powerful for laying out content on a page. However, it is pretty limited in terms of style. Removing of formatting and also changing formatting can be a tedious task because HTML formatting is done via tags. For these things CSS comes in very handy, and it is what we will be learning next. Just like HTML, it can easily be controlled via JavaScript.

3.4 CSS

To further segregate the document's presentation and content, CSS comes very handy. CSS (abbreviation for cascading style sheets) is a powerful way to control the way a web page looks, and give it that dynamic feel without actually resorting to coding.

CSS is very powerful in describing the style of documents, and defining complicated rules for the style that should be applied to a particular element and under what conditions. For example, if we wanted all links to be bold and green under normal conditions, and red and underlined when the mouse is hovered over it, the code would be as follows:

```
<style>
  <!--

  a {
      color: #00ff00;
      font-weight: bold;
  }

  a:hover {
      color: #ff0000;
      text-decoration: underline;
  }
  -->
  </style>
```

This would be placed inside the document header.

Basically, all styles are placed within a `<style>` tag, enclosed within the comment tag `<!-- -->`. They can also be placed in a CSS file, and included within the HTML page. To include a CSS file or stylesheet (known as my_style.css) in an HTML file, we use the `<link>` tag in the header of the document. The syntax to do so is:

```
<link rel="stylesheet" type="text/css" href="my_style.css"/>
```

In CSS, to define any sort of style, you need to have two things ready. First, what all elements it should apply to, i.e. the scope of the style, also known as the selector. Second, you need to provide what styles you want to apply to it. This is similar to the situation where you need to assign a particular value to a variable. Here, you need the name of the variable as well as the value you want to assign.

However, unlike a case of assigning a value to a variable, we are applying a set of styles to a group of elements. This is a group assignment rather than a one-to-one correlation.

First, we need to select what parts of the page we want to apply these rules. For this, we have to come up with a selector, or a common pattern that all the elements that need to be styled that way have.

To simply apply a style to all elements of a particular tag, as in the example shown before, the selector is just the name of the tag:

```
a {
    color: #00ff00;
    font-weight: bold;
}
```

This would make all the text that appears inside an <a> tag (i.e. all links), green, and bold.

If you want to apply the same rule to multiple elements, you can separate those using commas. For example, if you also wanted all list elements and h1 elements to also appear green and bold, the code would be as follows:

```
a,h1,li  {
    color: #00ff00;
    font-weight: bold;
}
```

Often, we would not simply want to make everything in all instances of a tag look the same. In such cases, we can use the class selector. If you want to have different styles in different sections of the page, classes can come very handy. A class selector is defined using a class name preceded by a dot. In the following example, we'll define four different styles for four different sections of the page:

```
.normaltext {
        color: #000000;
        text-align: left;
        text-indent: 5px;
}
.specialtext {
        color: #00ff00;
        font-weight: bold;
}
.header {
        color: #00ffff;
        font-weight: bold;
```

```
        font-size: large;
        background-color: #000000;
    }
    .footer {
        color: #ffffff;
        font-size: small;;
        background-color: #000000;
    }
```

Here we have defined the following four classes:

- .normaltext: This class is defined for all the normal content text that will appear on the page. It is black in colour, aligned to the left, and has an indentation of 5 pixels.
- .specialtext: This class is for any special text we need to highlight. This could be an important sentence or word in a paragraph. It is green in colour, and bold.
- .header: This is for the content of the header of the page. It is cyan in colour, with a black background, using a large text size in boldface.
- .footer: This is for all the footer text. Usually copyright notices, etc. It will be white text in a black backdrop, and a small font size.

It is not sufficient to just define these classes. You need to specify where to apply them too. To do this, we add the 'class' attribute having the name of the class as its value to all the elements to which the style is applicable.

To apply the footer class style to the contents of a div element, do the following:

```
<div class="footer">
        Content of div element
    </div>
```

CSS supports some very advanced selectors. You can specify quite complicated rules for what style applies to which part of the document. Since CSS is a vast subject in itself, it is out of the scope of this book to go into details. However, most of the useful rules can be illustrated using the examples as follows:

Example 1:

To make all the text of all paragraph element (<p>) which

reside in elements green and bold, we can do:

```
div p {
        color: #00ff00;
        font-weight: bold;
    }
```

Example 2:

To make all the text green and bold in all paragraph elements (<p>) which have the class 'tree' modify the code as follows:

```
p.tree {
        color: #00ff00;
        font-weight: bold;
    }
```

Example 3:

To make all the text appear green when you hover the mouse over them, then use the following:

```
a:hover {
        color: #00ff00;
        font-weight: bold;
    }
```

Example 4:

To make all the text of all <a> links which link to the page **http://www.google.com** green, we do:

```
a[href="http://www.google.com"] {
        color: #00ff00;
        font-weight: bold;
    }
```

One very important thing about CSS styles is that they cascade, hence the name Cascading style sheets. What this means is that if you apply a style to a tag, and a different tag to the <p> elements inside it, with a different style to the <a> elements in that, the resultant style of the <a> element will be a combination of the styles applied to the , and the <p> and the <a> tags. In case of a conflict in styles, precedence is given in the following order:

i. They are overridden by external style sheet styles
ii. They are overridden by internal style sheet styles
iii. They are overridden by inline styles

The resultant styles are then applied to the element.

Now just to show you exactly how flexible CSS can be, and how you can achieve splendid effects using CSS alone, let us go back to our list, and format it using CSS.

We will leave the headings as they are, but we will make this into a menu, so that someone can click on a movie's name and go to its web site. For this, each movie has to be placed inside an `<a>` tag, with the URL of the web site in the `href` attribute.

Here is the new HTML code:

```
<html>
  <head>
    <title>My Favorite Movies</title>
     <link rel="stylesheet" type="text/css"
href="my_style.css" />
  </head>
  <body>
    <h1>A List of my favorite Movies</h1>
    <h2>Movie Series:</h2>
    <ul>
        <li><a href=".">The Matrix Series</
a></li>
     <li><a href=".">The Lord of the Rings
Series</a></li>
      <li><a href=".">Star Wars Series</a></
li>
    </ul>
    <h2>Individual Movies:</h2>
    <ul>
      <li><a href=".">Memento</a></li>
      <li><a href=".">Batman Begins</a></li>
       <li><a href=".">The Dark Knight</a></
li>
      <li><a href=".">Gattaca</a></li>
      <li><a href=".">Shawshank Redemption</
a></li>
    </ul>
  </body>
</html>
```

As you can see, little has changed except that each list element is now individually enclosed inside an `<a>` tag, and

the stylesheet is now included in the document. Enclosing the whole block of code within the <a> tag is one option. However, each movie has to link to a different page, and putting them in the same block wouldn't help. Also, even if we wanted to link them all to the same page, the results would differ from what we wanted — each link should individually behave like a button.

Linking them to '.' means that clicking on them will open the directory in which the file is placed.

Now for the CSS part:

```
li {
    margin: 5px 5px 5px 5px;
}

a {
    color: #ffffff;
    background-color: #232323;
    border: solid;
    border-width: thin;
    border-color: #0f0f0f;
    text-decoration: none;
    margin: 5px 5px 5px 5px;
    padding: 2px 2px 2px 2px;

}

a:hover {
    background-color: #4e4e4e;
    padding: 5px 5px 5px 5px;
}
```

The first style is applied to the <li> elements, and is just to give enough space around them make the button appear clearly.

The second style is for the anchor element <a>. It does the following:

- Sets the text colour to #FFFFFF or white
- Sets the background colour to #232323, which is a deep shade of grey.
- Tells the browser to draw a thin border around the text, of

colour `#0F0F0F`, which is a very dark shade of grey.

- Tells the browser not to display an underline, using the text-decoration style.
- Sets the margin around the border to 5 pixels on the top, right, bottom or left.
- Sets the padding around the text to 2 pixels on the top, right, bottom or left.

The third style is for the hover state of the `<a>` tag. When you move your mouse over the text, it uses the same styles with a slight change made to it. It does the following:

- Makes the background colour a lighter shade of grey.
- Expands the space around the text, to give the effect that the button is expanding.

This way, you can achieve powerful effects with just CSS alone, and since they can be controlled via JavaScript, there is no limit to what you can do with it. What we have touched here is just the tip of the iceberg. More on JavaScript follows.

JavaScript

4.1 Introduction to JavaScript

JavaScript is what drives the web and has defined the Web 2.0 experience. No one can deny the fact that it has made the web a much more interactive place.

JavaScript was created to make the web more interactive. Although it was introduced way back in the days of Netscape, it is now available on all browsers and has been standardised by ECMA as ECMAScript. By standardised, it means it will have the same syntax and effect on all browsers.

4.2 ...of browsers and DOM

The document object model (DOM) is the browser's internal representation of the HTML page. It defines how the browser and the developer 'sees' the page. One of the biggest problems encountered while creating JavaScript code is that different browsers represent the page differently, and thus the developer's job is to first find out which browser the code is running on, and then run the appropriate code written for that browser.

In the beginning, all browsers were continuously adding features in order to gain advantage, and this led to many non-standard HTML tags, and JavaScript procedures. Over time some of those have been accepted to become a standard, such as XMLHttpRequest while many others have now depreciated.

Since the internet is a large place, the vast variety of system configurations make the developer's job all the more difficult. Even though most browsers today are trying to accommodate web standards, there is still a plethora of computer users who are using older browsers.

You can ask all those you know to upgrade your browsers; however, updating the online community is not possible.

Once you push a new browser, it will take some time, may be even years, for it to represent a majority of internet users. Even now, there are people using Firefox 1.5 or Internet Explorer 6.

Even standards take time to sink in, because despite the latest browsers claiming to be fully standards-compliant (which they are not), we still have the problem that everyone will not have the latest browser.

So between programming across different browsers, and taking into account the different versions of those browsers, your code can become very complicated and bulky if you want to make it work for everyone. You must also ensure that a user using a browser that doesn't support JavaScript or one in which JavaScript is disabled, is not left stranded.

4.3 Basic JavaScript

JavaScript can be placed anywhere in your HTML document as long as it is enclosed within a `<script>` tag, which can be placed anywhere within the body, or in the header. If the code needs to be reused, it can also be placed in an external file which in referenced in the document, via the `<script>` tag's 'src' property. Since the header of the document is loaded before the page starts displaying, placing the script in the header ensures that it is loaded before the page can call to any functions defined in it.

If any of you have used C, C++ or a similar language, you will find the syntax quite similar. The structure of the code is represented in pretty much the same way. The following is an example of some JavaScript code that takes two numbers, and finds the highest, lowest and average of the numbers using functions.

```
var a = 5;
var b = 16;
var greater = greaterof(a, b);
var lower = smallerof(a, b);
var average = averageof(a, b);

function greaterof(num1, num2){
     return num1 > num2 ? num1 : num2;
```

```
}

function smallerof(num1, num2){
     return num1 < num2 ? num1 : num2;
}

function averageof(num1, num2){
     return (num1 + num2) / 2;
}
```

Looks familiar? Even if it doesn't seem familiar, the syntax itself is quite apparent. You can declare variables using the keyword `var`, although it isn't necessary, and you can just write the variable name and equate it to its value.

Similar to most programming languages, JavaScript differentiates between equating and testing for equality. This means, if you need to assign the value `5` to a variable `num1`, you do the following:

```
num1 = 5;
```

However, if you need to check if the value of `num2` is `17` then:

```
num2 == 17;
```

Those familiar with programming will be wondering, what is the point of all of this if you are not outputing anything. Since JavaScript is used with HTML, there is no output in the traditional sense. What you will be doing instead, is writing this content to the page in some or the other element, or merely manipulating the layout as a result.

For those of familiar with C, Java or similar languages, you will notice the similar syntax, but for those of you who aren't, let's go through the basics of the language:

Performing arithmetic:

Operation	Description	Use
+	Addition	z = x + y
-	Subtraction	z = x – y
*	Multiplication	z = x * y
/	Division	z = x / y
%	Modulus i.e. Remainder	z = x % y
++	Increment	z++
--	Decrement	z--

You can perform compound operations using the operators, +=, -=, *=, /=, %=. These are used to perform an operation on a variable, and store the result in the original variable. For example, if you need to increment the variable 'x' by '32', you would normally do it by writing 'x = x + 32', using compound operators you can do this simply as 'x += 32'.

Note that addition can also be performed on strings. Therefore, to add the strings "My name is" and "John Doe", you can simply write 'result = "My name is " + "John Doe"'.

Making comparisons:

Operation	Description	Use
==	Test for equality	4 == 4 is true 4 == '4' is true
!=	Test for inequality	4 == 5 is false
>	Greater than	5 > 4 is true 8 > 11 is false
<	Less than	4 < 5 is true 10 < 8 is false
>=	Greater or equal	4 >= 4 is true
<=	Less or equal	11 <= 11 is true
===	Strictly equal	4 === 4 is true 't' === 't' is true 4 === '4' is false

Logical operations:

Operation	Description	Use
&&	And	a && b is true if both a and b are true
||	Or	a || b is true if at least one of a and b are true
!	Not	!a simply negates a such that !a is true if a is false and it is false if a is true

Branching:

Branching is needed when the outcome of the program depends on a particular condition. For example, if you needed

to set a variable 'x' to '32' if 'y' is greater that '5' and to '64' otherwise, you would write this as:

```
if (y > 5) {
     x = 32;
}
else {
     x = 64;
}
```

This can be represented in a much more compact notation as:

```
x = (y > 5) ? 32 : 64;
```

This achieves the same results as before but is much more compact. However, if you use the 'if' statement, you can do more than just assign values, any type of complicated code can be inserted within the blocks after 'if' or 'else'.

The syntax of this operator is;

variable = (condition) ? (value if true): (value if false)

You can also chain multiple if... else statements one after one another to perform different results for different conditions. However, for many situations, you can instead use switch... case.

Here is an example, of how you can use if for selecting pizzas:

```
switch (pizza_size) {
     case 'small':
          size = 10;
          break;

     case 'normal':
          size = 14;
          break;

     case 'large':
          size = 16;
          break;
}
```

This small example will set the 'size' variable to the size of the pizza in inches, if the 'pizza_size' variable is set to one of 'small', 'normal', and 'large'.

Looping:

With JavaScript you can use two types of loops, the 'for' loop and the 'while' loop. They can both do the same thing, but in different ways. The 'for' loop is more suited when one needs to iterate over fixed values, such as looping over all the paragraphs in a document and setting them to bold.

Let us see an example where we use the for loop to add the first 100 numbers:

```
sum = 0;
for (i = 1; i =< 100; i++) {
     sum += i;
}
```

The 'for' loop has three parts in the bracket:

- In the first part, you set any variables you need to, this is performed only once at the beginning of the loop.
- In the second part, you test the condition for the execution of the loop, this is tested before the loop is executed, so if the statement is false to begin with, the loop will never execute.
- The third part performs operations that change the counter, incrementing or decrementing it.
- The statements in the loop are executed every time the loop is run.

Now let's see the 'while' loop doing the same:

```
while (cond==true)
{
     ...statements...
}
```

Doing the same using the for loop would be unnecessarily confusing. The 'while' loop shows its strengths here.

Functions:

Functions are a feature present in almost each and every programming language. For anyone who has studied mathematics, understanding functions is not conceptually difficult. However, practically, computer functions are different and more versatile. Let use see an example of a function which returns the square of an entered number:

```
function square(num){
    var num_square = num * num;
    return num_square;
}
```

A function basically has the following basic syntax:
function(paramater1, paramater2,...)
{
...statements...
}

If the function needs to return a result, it may do so by using the 'return' keyword, as seen in the example above. In the above example, we could also have directly returned 'num*num' instead of first storing it in a variable.

Datatypes:

JavaScript is a weakly typed language, meaning that variables do not have an implicit type at declaration. When you create a new variable its type depends on the value assigned to it. Conversion between types is also quite simple.

JavaScript has support for strings, dates, arrays, boolean values, and so on, natively. JavaScript also natively supports RegEx, but that is out of the scope of this book.

4.4 DOM

DOM is a standard that specifies how a browser is supposed to represent a web page internally, such that it can be manipulated using JavaScript. When we use JavaScript to change the appearance based on human interaction, DOM defines how we expect to detect the interaction, and how we can access and change the attribute of the document such that its appearance or behaviour is changed.

Since DOM is a general term, it is not specific to HTML only, but any document. In the HTML DOM, the main document is referred to using the object 'document', and this will be the thing we will use the most when working with JavaScript in web pages.

To manipulate an HTML page, we need to do three things — first, we need to find the element / part of the page that we

wish to modify. Second, we need to know how to access the property that we wish to modify. Finally, we need to know how to modify it.

This may seem trivial in stating, as it seems fairly obvious. However, JavaScript can sometimes be slightly counterintuitive, as it has been used as a playground for experimentation by different browsers in a bid for market share. Also, we need to be able to trigger this change based on some event that occurs. For example, a user clicking on text.

First, one of the easiest ways to access any element on the page is using it's 'id' attribute. 'id' is similar to variable names — they are used to refer to an element in particular, and are be reused or repeated. So while desigining the HTML page, we need to ensure that any elements we wish to manipulate have unique ID's assigned to them, so that we may use them in the JavaScript code.

The 'document' object contains references to all the items that are part of the HTML page. Using the 'document' object you can access elements in the following ways:

- document.getElementById('element_id')
  This method will return a reference to the element on the page which has an id of 'element_id'.
- document.getElementByName('element_name')
  This method will return an array of elements with the name 'element_name'
- document.getElementByTagName('tag_name')
  This method will return an array of all the tags named 'tag_name'

Now that you have a reference to the object or objects you wish to manipulate, the next step is to access these properties. Attributes of the element can be directly accesed by using the '.' operator. That is, to change the color of the background of the page, you would set the objects 'bgColor' property to whatever the new color is required to be. Styles associated with the element can be changed using the 'style' property of the object. In HTML, use lower case for all tags and attributes, while in JavaScript we use camelCase. Also, CSS supports using the hyphen to separate words in style names. However, JavaScript doesn't support this feature. In JavaScript, the hyphen denotes subtraction.

Therefore, all styles and attributes are to be converted to camelCase notations.

Now for some information on what camelCase is. While programming it is always best, to provide meaningful names for variables. It is better to name a variable denoting simple interest on an amount as 'interest' rather than 'i'. It is even better if a convention is adopted for naming variables such that anyone can figure out what each variable is responsible for storing at any given point. There are many variable naming conventions that have been adopted by many different organisations. Some of the popular ways are (in case of multiple word variables), to have words separated by underscores ('_'), or by capitalising the first alphabet of the next word. Therefore, if you wished to store the average length of different rivers, you could name the variable as 'average_length_rivers' by the first convention or 'averageLengthRivers' by the second. The second convention is known as the camelCase convention because it resembles a camel with a hump.

So a style called 'background-color' when accessed through JavaScript will have to be done as 'element.style.backgroundColor'. So now you can get to an element, and access it. All you need to do is change the value. Changing is as simple as assigning a new value! So now we are ready to do some real stuff!

4.5 Learning by example

Now let's get to the fun coding part! Here we will work with actual code, and see what it does. The best way to learn is by example, and the best way to progress it to play with things rather than work. So let's play!

We will now go back, back to the example we used while learning HTML. And here it is:

```
<html>
  <head>
    <title>My Favorite Movies</title>
  </head>
  <body>
    <h1>A List of my favorite Movies</h1>
```

```
    <h2>Movie Series:</h2>
    <ul>
       <li><a href=".">The Matrix Series</a></
li>
        <li><a href=".">The Lord of the Rings
Series</a></li>
        <li><a href=".">Star Wars Series</a></
li>
    </ul>
    <h2>Individual Movies:</h2>
    <ul>
      <li><a href=".">Memento</a></li>
      <li><a href=".">Batman Begins</a></li>
      <li><a href=".">The Dark Knight</a></li>
      <li><a href=".">Gattaca</a></li>
        <li><a href=".">Shawshank Redemption</
a></li>
    </ul>
  </body>
</html>
```

What we will do now is to associate an ID with some of the elements so that they may be manipulated. We will associate an ID of 'sermovies' with the list of movie series, and an ID of 'indmovies' with the individual movies. So finally it should look like this:

```
<html>
  <head>
    <title>My Favorite Movies</title>
  </head>
  <body>
    <h1>A List of my favorite Movies</h1>
    <h2>Movie Series:</h2>
    <ul id="sermovies">
       <li><a href=".">The Matrix Series</a></
li>
        <li><a href=".">The Lord of the Rings
Series</a></li>
        <li><a href=".">Star Wars Series</a></
li>
```

```
    </ul>
    <h2>Individual Movies:</h2>
    <ul id="indmovies">
      <li><a href=".">Memento</a></li>
      <li><a href=".">Batman Begins</a></li>
      <li><a href=".">The Dark Knight</a></li>
      <li><a href=".">Gattaca</a></li>
        <li><a href=".">Shawshank Redemption</
a></li>
    </ul>
  </body>
</html>
```

Now how about we add some titles? We will first create a function, which will add a movie to the movie series list. Let's call this function 'addmovie'.

Here it is:

```
<script language="JavaScript">
  <!--
  function addmovie(){
       document.getElementById('sermovies').
innerHTML
  += '<li><a href=".">The Terminator Series</
a></li>';
  }
  -->
</script>
```

We place this in the head segement of the page.

Now what we need to do is call this function as a result of something that the user does. For this example, how about we call it whenever the user clicks on the heading 'Movie series:'? To do this we will just add an attribute 'onclick="addmovie()"' to the `<h2>` tag containg the heading.

So finally we have:

```
<html>
  <head>
    <title>My Favorite Movies</title>
    <script language="JavaScript">
      <!--
```

```
        function addmovie(){
            document.getElementById('sermovies').
innerHTML
+= '<li><a href=".">The Terminator Series</a></
li>';
        }
        -->
    </script>
  </head>
  <body>
    <h1>A List of my favorite Movies</h1>
    <h2 onclick="addmovie()">Movie Series:</h2>
    <ul id="sermovies">
       <li><a href=".">The Matrix Series</a></
li>
        <li><a href=".">The Lord of the Rings
Series</a></li>
        <li><a href=".">Star Wars Series</a></
li>
    </ul>
    <h2>Individual Movies:</h2>
    <ul id="indmovies">
      <li><a href=".">Memento</a></li>
      <li><a href=".">Batman Begins</a></li>
      <li><a href=".">The Dark Knight</a></li>
      <li><a href=".">Gattaca</a></li>
        <li><a href=".">Shawshank Redemption</
a></li>
    </ul>
  </body>
</html>
```

Now when you click on the heading, you will see 'The Terminator Series' added to the 'Movie Series' list. Every time you click you will see another entry added to the list. The innerHTML property represents the HTML content within the tag. So anything you add to it, will go inside the `<ul>` tag. This is a very useful and powerful way of adding content to a page, and is quite simple.

Now let's make it a little more dynamic. Instead of adding the same entry over and over again, we will add a counter.

Every time you click, a new numbered entry called 'Series #' will be added, where # is the series no.

Let's see how this is done.

Now, what we added was a new variable called 'len', which will store the current no of items in the list. How it does that is, it retrieves the 'sermovies' list element, from which it retrieves an array all the <li> tag elements, and assigns the length of that to 'len'. So 'len' basically has the number of entries in the list. Now we increment 'len' to get the next entry's number. This is now added in front of the new entry, which is then appended to the list.

The end result — every time you click on the heading, a new series keeps getting added to the list. They will come numbered as 'Series 4', 'Series 5' and 'Series 6'. The reason they start from 4 is that 3 items are already there in the list, 'The Matrix Series', 'The Lord of the Rings Series' and 'Star Wars Series'.

Surely, this is more dynamic than before, but still, it is essentially useless! Let us now make it more interactive by allowing a user to enter the name of the movie series to be added.

For this we will need to add some content to the HTML page, some kind of form where they can enter a movie name. Let us place a text input box under the movie series list for the purpose, and add a button to add a name to the list. Here is the final HTML code for that:

```
<html>
  <head>
    <title>My Favorite Movies</title>
    <script language="JavaScript">
      <!--
      function addmovie(){
           movieName = document.getElementById(
'newseries').value;
   document.getElementById('newseries').value =
'';
          document.getElementById('sermovies').
innerHTML += '<li><a href=".">'
+ movieName +
'</a></li>';
```

```
        }
        -->
    </script>
  </head>
  <body>
    <h1>A List of my favorite Movies</h1>
    <h2>Movie Series:</h2>
    <ul id="sermovies">
       <li><a href=".">The Matrix Series</a></
li>
        <li><a href=".">The Lord of the Rings
Series</a></li>
        <li><a href=".">Star Wars Series</a></
li>
    </ul>
     <input id="newseries" type="text" /><input
type="button"  onclick="addmovie()"  value="Add
Series" />
       <h2>Individual Movies:</h2>
    <ul id="indmovies">
      <li><a href=".">Memento</a></li>
      <li><a href=".">Batman Begins</a></li>
      <li><a href=".">The Dark Knight</a></li>
      <li><a href=".">Gattaca</a></li>
        <li><a href=".">Shawshank Redemption</
a></li>
    </ul>
  </body>
</html>
```

Now let's see what we've done here. First, we've added two new `<input>` tags below the first list. One is of type 'text', and the other 'button'.

The `<input>` tag of type 'text' is a text input box, where the user may type the name of the movie. It has been given an ID of 'newseries', so that its value can easily be used in JavaScript later on.

The `<input>` tag of type 'button' is well, a button! It has an 'onclick' property set to 'addmovie()', so that when the user clicks on the button, the 'addmovie' function is called. It also

has an attribute 'value', set to 'Add Series', this is the label that will appear on the button.

The addmovie function itself has been modified to accommodate this change. The changes made to the function are:

- A new variable called 'movieName' is now added and set to the value of the textbox with an 'id' of 'newseries', in the line: `movieName = document.getElementById('newseries').value;`
- The value of the textbox is being cleared by setting it to an empty string. This way the user can start typing a new name immediately. We do this using: `document.getElementById('newseries').value = '';`
- The value being set for each new entry to the list is set to the value entered in the textbox. This is done by modifying the old code which would just add a number after the string 'Series ' to: `document.getElementById('sermovies').innerHTML += '<li><a href=".">' + movieName + '</a></li>';`
- The old code to calculate the number of items in the list has been removed.

If you run it now you will see that the value you enter in the textbox is added to the list when you click the button.

Now let's play around a little more. We will now add another set of textbox and button to enter movies in the individual movies list.

Also the function 'addmovie' is too restricted, as it will only add movies to the 'Movie Series' list. We will now give it some parameters so that it can work with both cases, and infact can be used any any new list we might add.

Here is the new proposed function:

```
<script language="JavaScript">
<!--
        function addmovie(nameid, listid){
movieName = document.getElementById(nameid).
value;
document.getElementById(nameid).value = '';
document.getElementById(listid).innerHTML += 
'<li><a href=".">'
+ movieName + '</a></li>';
  }
```

```
    -->
    </script>
```

The function now supports two parameters — one for the textbox to get the movie name, and the other is the ID of the list to which it is to be added.

Now to the HTML code we will add another textbox and button below the movie list. Also, now we have to update the 'onclick' attribute of the button so that it uses the function 'addmovie' with its new syntax.

Finally we get this:

```
<html>
  <head>
    <title>My Favorite Movies</title>
    <script language="JavaScript">
      <!--
      function addmovie(nameid, listid){
                    movieName = document.
getElementById(nameid).value;
            document.getElementById(nameid).
value = '';
            document.getElementById(listid).
innerHTML += '<li><a href=".">' + movieName +
'</a></li>';
        }

        -->
    </script>
  </head>
  <body>
    <h1>A List of my favorite Movies</h1>
    <h2>Movie Series:</h2>
    <ul id="sermovies">
        <li><a href=".">The Matrix Series</
a></li>
      <li><a href=".">The Lord of the Rings
Series</a></li>
        <li><a href=".">Star Wars Series</a></
li>
    </ul>
    <input id="newseries" type="text" />
```

```
      <input type="button" onclick="addmovie('n
ewseries','sermovies')" value="Add Series" />
       <h2>Individual Movies:</h2>
       <ul id="indmovies">
         <li><a href=".">Memento</a></li>
         <li><a href=".">Batman Begins</a></li>
          <li><a href=".">The Dark Knight</a></
li>
         <li><a href=".">Gattaca</a></li>
         <li><a href=".">Shawshank Redemption</
a></li>
       </ul>
       <input id="newmovie" type="text" />
      <input type="button" onclick="addmovie('n
ewmovie','indmovies')" value="Add Movie" />
    </body>
  </html>
```

You should pretty much have a grasp on basic JavaScript by now and discover just how much you can change. However, just play around just a little bit more with this before giving up on it.

The next change we will make will be to remove one of the textbox and use the other for adding entries to both the lists. We will also modify the function as it now needs only one parameter, the ID of the list in which to add the entry.

We won't go into how it works, you will probably be able to understand that by yourself by now. Here is what we get:

```
  <html>
    <head>
      <title>My Favorite Movies</title>
      <script language="JavaScript">
        <!--
        function addmovie(listid){
            movieName = document.getElementByI
d('newmovie').value;
          document.getElementById('newmovie').
value = '';
            document.getElementById(listid).
innerHTML += '<li><a href=".">'
```

```
    + movieName +
    '</a></li>';
          }
          -->
        </script>
      </head>
      <body>
        <h1>A List of my favorite Movies</h1>
        <h2>Movie Series:</h2>
        <ul id="sermovies">
            <li><a href=".">The Matrix Series</
a></li>
          <li><a href=".">The Lord of the Rings
Series</a></li>
          <li><a href=".">Star Wars Series</a></
li>
        </ul>
        <h2>Individual Movies:</h2>
        <ul id="indmovies">
          <li><a href=".">Memento</a></li>
          <li><a href=".">Batman Begins</a></li>
           <li><a href=".">The Dark Knight</a></
li>
          <li><a href=".">Gattaca</a></li>
          <li><a href=".">Shawshank Redemption</
a></li>
        </ul>
        <input id="newmovie" type="text" />
        <input type="button" onclick="addmovie('
indmovies')" value="Add Movie" />
        <input type="button" onclick="addmovie('
sermovies')" value="Add Series" />
      </body>
    </html>
```

Now, instead of having two text input boxes for two different lists, we have one textbox, the contents which are added to different lists depending on the button clicked.

Just to show you how we can control styles in JavaScript, we will add a quite common feature found on many sites, a collapsible panel. In a collapsible panel, when you click on the

title of a widget, it hides the rest of the content. Like the chat windows which appear in Gmail, they can be minimised, or 'collapsed' to show just the name of the person you are chatting with.

In the next example, what we'll do is add a function to collapse a list, with a parameter which asks for which list to collapse. We will call this function whenever someone clicks on the headings.

Here is what we get:

```
<html>
  <head>
    <title>My Favorite Movies</title>
     <link rel="stylesheet" href="new_file.
css">
    <script language="JavaScript">
      <!--
      function addmovie(listid){
          movieName = document.getElementByI
d('newmovie').value;
        document.getElementById('newmovie').
value = '';
            document.getElementById(listid).
innerHTML += '<li><a href=".">'
  + movieName +
  '</a></li>';
      }

      function collapse(listid){
                  listObject = document.
getElementById(listid);
            if (listObject.style.display ==
'none') {
                  listObject.style.display =
'';
          }
          else {
                  listObject.style.display =
'none';
          }
      }
      -->
```

```
        </script>
      </head>
      <body>
        <h1>A List of my favorite Movies</h1>
       <h2 onclick="collapse('sermovies')">Movie
Series:</h2>
        <ul id="sermovies">
            <li><a href=".">The Matrix Series</
a></li>
          <li><a href=".">The Lord of the Rings
Series</a></li>
       <li><a href=".">Star Wars Series</a></li>
        </ul>
        <h2 onclick="collapse('indmovies')">Indi
vidual Movies:</h2>
        <ul id="indmovies">
          <li><a href=".">Memento</a></li>
          <li><a href=".">Batman Begins</a></li>
        <li><a href=".">The Dark Knight</a></li>
          <li><a href=".">Gattaca</a></li>
          <li><a href=".">Shawshank Redemption</
a></li>
        </ul>
        <input id="newmovie" type="text" />
        <input type="button" onclick="addmovie('
indmovies')" value="Add Movie" />
        <input type="button" onclick="addmovie('
sermovies')" value="Add Series" />
      </body>
    </html>
```

In this example, you can expand or collapse the lists by clicking on their respective headers. This is done by toggling the value of their 'display' style between 'none' and the default value.

These are the basics of JavaScript. The different options available to you when working with the different elements on the page are just too many to enumerate or remember. By using a reference source for such things, you already create wonderful JavaScript applications. Next we will begin with Ajax. Since it is just a way of using JavaScript, it will merely be a deeper look into some of JavaScript's functionality.

AJAX

5.1 The spirit of AJAX

The basic spirit of AJAX is in making best use of technologies we already have, to make the web more responsive and interactive. All you need is a browser with JavaScript support which is essentially every browser out there. Unless someone has disabled JavaScript or is using a really old browser, you can be sure that your web site will work. However, this doesn't mean web sites no longer need to work towards ensuring compatibility with all browsers.

The internet is no longer a web of documents. It has now become an environment of inter-connected services. Your work is now independent of your desktop environment. Whether you're using Linux, Windows, or any other OS, all you need is a modern browser, and you can do the rest online.

Not convinced? Let's see, what all you can do online these days:

Store documents and files at: **box.net**, DropBoks, **acrobat.com**, Google Docs, etc.

Edit Documents at: Google Docs, buzzword, **zoho.com**, etc.

Store images at: Flickr, Photobucket, Picasa Web Albums, etc.

Edit Images at: **photoshop.com**, **picnik.com**, etc.

The list goes on...

If you search online you will quickly find an online replacement for almost everything you do offline. In fact these services are now open in such a way, that you can store your documents at **box.net** and open them at **zoho.com** without downloading them! Or store images at Picasa or Flickr and edit them using **photoshop.com**.

Today, entire operating systems are available online, for example GlideOS and eyeOS. These are online environments which allow you to do virtually everything, online. Instead of making your computer more and more portable, you can just make your environment more and more portable. No pen

drives, no laptops, the only thing you need wherever you go is the internet, and you can start working.

Guess what each and every one of them has in common? AJAX! AJAX in a way, is the spirit of the web.

Often times you will notice that some AJAX-based web sites take a long time to load, and this has been often cited as a big flaw, however the fact is that once the web site loads, it covers up for lost time pretty quick. If you use Gmail, try comparing its working in 'Standard' mode and 'basic HTML' mode. You will realise that although it loads much faster in 'basic HTML' mode, it takes much longer to open an email, switch folders etc as the page is refreshed every time. Functionality such as chatting in the browser, Web Clips etc. are also not available. The interface itself seems like a dull reminder of archaic times.

5.2 Starting AJAX coding

Since AJAX isn't really a language, there is no such thing as pure AJAX code. As such what we will learn now will be JavaScript code which uses the principles of AJAX.

You have learnt JavaScript, HTML, and XML in the previous sections, and here we will use all three. So if you tried to cheat by skipping straight to the juicy AJAXy bits of this book, beware! There is little chance you will learn anything here if you aren't already acquainted with those topics. Turn back now.

One of the features that is most used in AJAX is the XMLHttpRequest object, so let us begin with that. XMLHttpRequest is a JavaScript object that allows you to load data from a server at any time. Using XMLHttpRequest, you can be in control of when the data of the page is loaded instead of waiting for a user to click on a link. You can create an XMLHttpRequest object at any time, and use it to initiate a connection to a remote server. You can then load data from this server, and using JavaScript place it on your page after it has loaded.

The first hurdle we encounter while using XMLHttpRequest is browser compatibility. Quite a few people out there are still using older versions of Internet Explorer which require a spe-

cial way of creating the XMLHttpRequest. You can see how to detect the browser, and create the XMLHttpRequest object in the code example.

```
function createAJAXObject(){
var httpReq;
try {
httpReq = new XMLHttpRequest();
}
catch (e) {
try {
httpReq = new ActiveXObject("Msxml2.
XMLHTTP");
}
catch (e) {
try {
httpReq = new ActiveXObject("Microsoft.
XMLHTTP");
}
catch (e) {
    return false;
}
}
}
}
```

In this example, we are using the try...catch mechanism to create an AJAX object. The catch clause is executed if there is an error in the first attempt. Then it tries to create the object using ActiveX, as used by older versions of IE. If all else fails, it returns false indicating that AJAX will not work on the browser.

Now let us really get started, we will now develop a movie database, you can download the sample data from the Digit web site, or create your own. The sample data contains a list of the top 100 earning movies. The data is out of date; however it is suitable for our purposes. Download and play the file in the same directory as the one in which you place the HTML file you create.

To truly make this an AJAX application, we will have to do the following:

- Load an external XML file containing the data
- Create an HTML representation of the data
- Place it in the desired location in the document

Before proceeding with the JavaScript code, let us first create the basic structure of the HTML document that will contain our application. We will have a simple HTML page with a single table with three columns — one column for the movie's rank, another for the movie's name, and the last for its earning.

```
<html>
    <head>
        <title>My AJAX App</title>
    </head>

    <body>
    <table width="500" border="1" id="movies">
            <tr>
                <th scope="col">Rank</th>

                <th scope="col">Name</th>

                <th scope="col">Earning</th>
            </tr>
        </table>
    </body>
</html>
```

When you open this in the browser, it will appear as a table with three cells, for the titles of each column. Here you are presented with some, perhaps, unfamiliar HTML tags. The `<th>` tag is for table headers, and its scope attribute defines that it is a header for a column. Here, we could have just used normal `<td>` cells, but since HTML provides a facility to demarcate the purpose of the data, it is best to use it. It can also be beneficial to search engines and while applying styles.

In our first example, we will just make the application load the XML data, and add it on the page as entries in the table. The length and complexity of the code may come as a shock; however keep in mind that we are starting of with a fairly complicated example. It will also make much more sense once we go through how it works, and what's going on.

```
<html>
```

```
    <head>
        <title>My AJAX App</title>
<script type="text/javascript">
            <!--
            var dataLoader;
            var data;

            function createAJAXObject(){
                var httpReq;
                try {
              httpReq = new XMLHttpRequest();
                }
                catch (e) {
                    try {
 httpReq = new ActiveXObject("Msxml2.XMLHTTP");
                    }
                    catch (e) {
                        try {
                                httpReq = new
ActiveXObject("Microsoft.XMLHTTP");
                        }
                        catch (e) {
                            return false;
                        }
                    }
                }
                return httpReq;
            }

            function loadData(){
            dataLoader = createAJAXObject();
                 dataLoader.onreadystatechange
= onDataLoad;
               dataLoader.open('GET', 'movies.
xml', true);
                dataLoader.send(null);
            }

            function onDataLoad(){
              if (dataLoader.readyState != 4)
```

```
                    return;
                if (dataLoader.status != 200 &&
dataLoader.status != 0)
                    return;

                data = dataLoader.responseXML.
documentElement.getElementsByTagName('movie');
                movietable = document.getEleme
ntById('movies');

            for (i = 0; i < data.length; i++) {
              newrow = movietable.insertRow(-1);

          newrow.insertCell(-1).innerHTML =
data[i].getElementsByTagName('rank')[0].text-
Content;

        newrow.insertCell(-1).innerHTML =
data[i].getElementsByTagName('name')[0].text-
Content;

       newrow.insertCell(-1).innerHTML =
data[i].getElementsByTagName('earning')[0].
textContent;
                }
            }
            -->
</script>
    </head>
    <body onload="loadData()">
                <table  width="500"  border="1"
id="movies">
            <tr>
                <th scope="col">Rank</th>
                <th scope="col">Name</th>
                <th scope="col">Earning</th>
            </tr>
        </table>
    </body>
</html>
```

Rank	Name	Earning
1	Titanic	600788188
2	STAR WARS (1977)	460988007
3	Star Wars: Episode I: The Phantom Menace	431088295
4	E.T. THE EXTRA-TERRESTRIAL (1982)	399804539
5	Jurassic Park	357067947
6	FORREST GUMP (1994)	329694499
7	Lion King, The	312855561
8	Return of the Jedi	309153948
9	Independence Day	306169255
10	Sixth Sense, The	293501675
11	Empire Strikes Back, The	290266497
12	Home Alone	285761243
13	JAWS (1975)	260000000
14	Batman	251188924
15	Men in Black	250690539
16	Toy Story 2	245823397
17	RAIDERS OF THE LOST ARK (1981)	245034358
18	Twister	241708908
19	Ghostbusters	238600000
20	Beverly Hills Cop	234760478

Your first AJAX

When you run this in the browser, you will notice a list of 100 movies in the table. You can tell that these have been added dynamically from the XML file, since they weren't in the document to begin with. If you don't see any data, make sure you downloaded the sample XML file from the ThinkDigit web site, and placed it in the same directory. The file should be called 'movies.xml'. Feel free to examine the XML file in any text editor, you will be able to see how the data is stored in it, and can add your own entries or modify the ones already there.

Take a look at the XML file so you know what it contains; only the first three entries are shown here as the complete file is quite large.

```
<movies>
    <movie>
        <rank>1</rank>
        <name>Titanic</name>
        <earning>600788188</earning>
    </movie>
    <movie>
        <rank>2</rank>
        <name>STAR WARS (1977)</name>
        <earning>460988007</earning>
    </movie>
    <movie>
        <rank>3</rank>
        <name>Star Wars: Episode I: The
Phantom Menace</name>
```

```
            <earning>431088295</earning>
        </movie>
        <movie>
            <rank>4</rank>
            <name>E.T.  THE  EXTRA-TERRESTRIAL
(1982)</name>
            <earning>399804539</earning>
        </movie>
    </movies>
```

The code seems complicated doesn't it? To make it simpler let's focus on what needed to be done and how it was done.

First, we need to load the external XML document containing the data. In this case, we make the data load when the page loads, using the 'onload' event of the `<body>` tag. To load the data, we first need to create an XMLHttpRequest object, which we create using the 'createAJAXObject' function described earlier. To actually use the object, we need to do the following:

- We need to provide a function that will be called when the data finishes loading, or fails to load. We do this by setting the 'onreadystatechange' property of the XMLHttpRequest object to a function which will handle the data, or error in loading.
- We need to supply the target file to load, and the mode in which it is to be loaded. This is done by calling the 'open' function of the XMLHttpRequest object. The function takes three parameters. The first is the mode, usually this will be 'GET', to retrieve data from a location. The second parameter specifies the path of the file to be loaded, in this case 'movies.xml', as the file is in the same directory as the HTML file. The final parameter is specifies whether we wish to load the data asynchronously. Loading synchronously means that the execution of the code will stop till the entire document located at the remote site has downloaded, and with asynchronous loading, the data will load in the back while the file continues executing. We wish to use it asynchronously, so we set it to true.
- Finally, we inject this data into our page as done in the 'onDataLoad' function.

The 'onDataLoad' function deserves an explanation owing to its complexity. Let us look at it step by step:

- `if (dataLoader.readyState != 4)`
  `return;`
  This code is responsible for checking if the data has finished loading. A ready state of 4 indicates that the request is complete.
- `if (dataLoader.status != 200 && dataLoader.status != 0)`
  `return;`
  This code checks whether the data loaded successfully. A status of 200 means that the data was loaded successfully. However status messages are not sent when loading a local file, such as we are using, and the example can fail to work, so we need to add a check for a status of '0' too. The statement inside the brackets after 'if' is a check for status NOT equal to '200' AND status NOT equal to 0. That is to say it should be neither '200' nor '0', or else the function will 'return' without doing anything.
- `data = dataLoader.responseXML.documentElement.getElementsByTagName('movie');`
  Here we assign the list of movie elements in the loaded document to a variable called 'data'.
- `movietable = document.getElementById('movies');`
  Here we assign a reference to the table containing the movie list to movietable, so that it may be conveniently used, or called.
- `for (i = 0; i < data.length; i++) {`
  This loop will run for all the items in the XML data source.
- `newrow = movietable.insertRow(-1);`
  This will add a new row into the movie table, and put a reference to the newly created row in the variable 'newrow'.
- `newrow.insertCell(-1).innerHTML = data[i].getElementsByTagName('rank')[0].textContent;`
  `newrow.insertCell(-1).innerHTML = data[i].getElementsByTagName('name')[0].textContent;`
  `newrow.insertCell(-1).innerHTML = data[i].get`

`ElementsByTagName('earning')[0].textContent;`

These three lines add the values for the three columns. They work in the same manner, so let us look at only the first. Here 'data[i]' refers to the i-th element of data in the data source. The `getElementsByTagName` function is used on the i-th data element, to retrieve a list of elements in that node which are of the type 'rank'. Since there will be only one, we use the 0-th element, which will give us the node containing the rank of the movie. We then retrieve the contents of the tag, which is then assigned to a newly inserted cell in the table using the innerHTML property.

Now that you've been introduced to the developer side of things, let's look behind the scenes. How does this application flow?

- The browser loads the main HTML page
- The browser calls the JavaScript function 'loadData'
- The browser sends a request to load 'movies.xml'
- The browser calls the function 'onDataLoad' every time the status of its request for movies.xml changes, i.e. when the request is initiated, when the request is sent, when it is processed, and finally when the request is complete.
- When the status becomes '4', implying that the request is complete and loaded, the browser starts executing the rest of the 'onDataLoad' function.
- The 'onDataLoad' function iterates over each data entry from the XML file and adds them to the page in the table.

That should cover it. We hope you've grasped the concept of what is going on in this example. If you are in doubt, go through it again, it will greatly help in understanding what's to come.

5.3 Further down the AJAX road

In this section, we will continue with the AJAX application we created above, and continually improve on it, making it into a true database.

The application we constructed before was a fair example, but it's quite static. Looking at it, one cannot discern it from a

normal HTML file already containing the data. So now we will get to the part where we can truly see the advantages of AJAX.

In this step, we will add pagination, which is to divide the data into pages, and display one at a time. To do this, we need to control the loop that adds data to the page, and rather display a subset of the whole data. Also, we need to clear the old data before adding new data.

We also need to add a navigation menu so the user can go to any page he wants, and we need to add functions to support this.

Let us first calculate and store some of the data that will later be repeatedly used. For example, the number of data items, items per page, and number of pages.

For this, we will add variables to our script outside of any function, so that they may be accessible by all functions.

```
var dataLoader;
var data;
var dataLength = 0;
var itemsPerPage = 10;
var noOfPages = 0;
var movieTable;
```

The names of the variables make their purpose quite clear, it involves more typing, but you can come back years later and still understand what you did. Note that we have set the value of 'itemsPerPage' to 10, we can set this to any value, but for now this is a reasonable amount.

We will also now split the 'onDataLoadFunction' and put the code that adds data on the page to a separate function. The 'onDataLoadFunction' will now just be responsible for initialising the data. It will calculate and store the length of the data and the number of pages into their respective variables, after which it will call the new function 'displayPage' to actually add the data to the page.

```
function onDataLoad(){
    if (dataLoader.readyState != 4)
        return;
    if (dataLoader.status != 200 && dataLoader.status != 0)
        return;
    data = dataLoader.responseXML.documen-
```

```
tElement.getElementsByTagName('movie');
        movieTable = document.getElementById('m
ovies');
        dataLength = data.length;
        noOfPages = data.length / itemsPerPage;
        displayPage(0);
```

The function is quite simple; the only new thing it does is set 'dataLength' to the number of items in the XML database. It also calculates the number of pages of data by dividing the number of items of data by the items to be shown on each page. It calls the 'displayPage' function with a parameter of '0', basically telling it to display the first page.

The 'displayPage' function is just a modified second-half of the original 'onDataLoad' function. Instead of displaying the whole list, it calculates the starting and ending positions of the items on the supplied page number, and displays only those.

Another thing required of this function is to clear out old data. We do that by repetitively deleting the second row of the table (index 1) till there is only one row left, which is the header. As you may have guessed this is done using the while loop in the code example. It keeps deleting the second row ('movieTable.deleteRow(1)') as long as the no. of rows ('movietable.rows.length') is more than one.

```
function displayPage(pagenum){

        startItem = pagenum*itemsPerPage;
        endItem = startItem + itemsPerPage;

        while( movieTable.rows.length >1 )
               movieTable.deleteRow(1);

        for (i = startItem; i < endItem; i++) {
             newrow = movieTable.insertRow(-1);
               newrow.insertCell(-1).innerHTML =
data[i].getElementsByTagName('rank')[0].text-
Content;
               newrow.insertCell(-1).innerHTML =
data[i].getElementsByTagName('name')[0].text-
Content;
```

```
                newrow.insertCell(-1).innerHTML
=   data[i].getElementsByTagName('earning')[0].
textContent;
        }
    }
```

With these modifications alone, your page will display the first page on loading, which will list the movies ranked from 1 to 10.

Let us now add a navigation menu, which will take us to the page we want. For this, we need to modify the HTML part of the page. We will add a table below the data, which will show the page numbers. This will be a simple table with one row.

```
<table id="navMenu" border="1">
    <tr>

    </tr>
</table>
```

We will also add a function which will populate this row with the different page numbers.

```
function buildNavMenu()
{
    navMenu = document.getElementById('navMe
nu').rows[0];

    for (i=0; i<noOfPages;i++){
        newcell = navMenu.insertCell(-1);
        newcell.innerHTML = i+1;
        newcell.onclick = new Function("d
isplayPage("+i+");");
    }
}
```

Let's examine what's going on in this function:

- `navMenu = document.getElementById('navMenu').rows[0];`

  In this line, we are storing a reference to the navigation menu row in the variable 'navMenu'. This essentially retrieves the 'navMenu' table, and retrieves a reference to its first row (index 0) and stores it in the variable 'nav-

Menu'.

- `for (i=0; i<noOfPages;i++)`
  Quite clearly, this will repeat the loop for each page.
- `newcell = navMenu.insertCell(-1);`
  This creates a new cell at the end of the row (index -1), and stores a reference to it in the variable 'newcell'.
- `newcell.innerHTML = i+1;`
  Now we add the page number inside the cell. The page number will be one more than the index, because while computers work with a starting index of 0, for humans (who will be using the page), 1 is a more comfortable place to start.
- `newcell.onclick = new Function("displayPage(" +i+");");`
  This line does the equivalent of setting the 'onclick' attribute of the table cell to 'displayPage(i)'. Let us look at the HTML equivalent of this when 'i = 3', for better clarity:
  `<td onclick='displayPage(3)'>4</td>`
  The code element 'new Function()' creates a function which has the code that is provided in the parameter.

Take a look at the complete code of our application as of now. The script part and the HTML part are shown separately, as the code is quite large.

Script part:

```
var dataLoader;
var data;
var dataLength = 0;
var itemsPerPage = 10;
var noOfPages = 0;
var movieTable;

function createAJAXObject(){
    var httpReq;
    try {
        httpReq = new XMLHttpRequest();
    }
    catch (e) {
        try {
         httpReq = new ActiveXObject("Msxml2.
```

```
XMLHTTP");
            }
            catch (e) {
                try {
            httpReq = new ActiveXObject("Microsoft.
XMLHTTP");
                }
                catch (e) {
                    return false;
                }
            }
        }
        return httpReq;
    }

    function loadData(){
        dataLoader = createAJAXObject();
     dataLoader.onreadystatechange = onDataLoad;
     dataLoader.open('GET', 'movies.xml', true);
        dataLoader.send(null);
    }

    function onDataLoad(){
        if (dataLoader.readyState != 4)
            return;

        if (dataLoader.status != 200 && dataLo-
ader.status != 0)
            return;

         data = dataLoader.responseXML.documen-
tElement.getElementsByTagName('movie');
        movieTable = document.getElementById('m
ovies');
        dataLength = data.length;
        noOfPages = data.length / itemsPerPage;
        displayPage(0);
        buildNavMenu();
    }
    function displayPage(pagenum){
```

```
        startItem = pagenum * itemsPerPage;
        endItem = startItem + itemsPerPage;

        while (movieTable.rows.length > 1)
            movieTable.deleteRow(1);

        for (i = startItem; i < endItem; i++) {
            newrow = movieTable.insertRow(-1);
              newrow.insertCell(-1).innerHTML =
data[i].getElementsByTagName('rank')[0].text-
Content;
              newrow.insertCell(-1).innerHTML =
data[i].getElementsByTagName('name')[0].text-
Content;
                newrow.insertCell(-1).innerHTML
=   data[i].getElementsByTagName('earning')[0].
textContent;
        }
    }

    function buildNavMenu(){
        navMenu = document.getElementById('navMe
nu').rows[0];

        for (i = 0; i < noOfPages; i++) {
            newcell = navMenu.insertCell(-1);
            newcell.innerHTML = i + 1;
                    newcell.onclick   =   new
Function("displayPage(" + i + ");");
        }
    }
```

HTML Part:

```
    <html>
        <head>
            <title>My AJAX App</title>
    <script type="text/javascript">
                <!--
                    SCRIPT CONTENT GOES HERE
```

```
                -->
    </script>
        </head>
        <body onload="loadData()">
                <table width="500" border="1"
id="movies">
                <tr>
                    <th scope="col">Rank</th>

                    <th scope="col">Name</th>

                   <th scope="col">Earning</th>
                </tr>
            </table>
            <table id="navMenu" border="1">
                <tr>
                    <td>
                    </td>
                </tr>
            </table>
        </body>
    </html>
```

To make it easier to further work on this, you can consider separating the JavaScript code and placing it in separate 'script.js' file, which can then be included in the HTML file using a script tag as follows:

Rank	Name	Earning
1	Titanic	600788188
2	STAR WARS (1977)	460988007
3	Star Wars: Episode I: The Phantom Menace	431088295
4	E.T. THE EXTRA-TERRESTRIAL (1982)	399804539
5	Jurassic Park	357067947
6	FORREST GUMP (1994)	329694499
7	Lion King, The	312855561
8	Return of the Jedi	309153948
9	Independence Day	306169255
10	Sixth Sense, The	293501675

1 2 3 4 5 6 7 8 9 10

AJAX example with navigation

```
    <script type="text/javascript" src="script.
js" />
```

You may have noticed the absence of 'Next' and 'Previous' page buttons, or of 'First' and 'Last' page buttons. So let's go ahead and add those. First, let's deal with the JavaScript part of this problem. We need to be able to do four things:

- Go to the first page: this is simple, all you need to do is call

'displayPage' with a parameter of '0'. No matter how many pages there are, 0 will always be the first page.

- Go to the last page: this becomes a little tricky, as we don't know which page will be the last till we read the data. What we will do instead is, extend the 'displayPage' function to support a parameter of '-1' which will always correspond to the last page. You can choose to use infinity, but that would take rather long to type!
- Go to the previous page and next page: We will create a new function for this, and also add a variable which stores the current page number. The function needs to account for the fact that the user may click on the next page button when they are on the last page already, or click on the previous page button when they are in fact on the first page. This new function will take a parameter of 1 or -1 to signify the direction in which to go.

Take a look at the updated 'displayPage' and new 'gotoPage' functions.

```
var currentPage = 0;

function displayPage(pagenum){

     if (pagenum == -1)
          pagenum = noOfPages - 1;

     currentPage = pagenum;
     startItem = pagenum * itemsPerPage;
     endItem = startItem + itemsPerPage;
     while (movieTable.rows.length > 1)
          movieTable.deleteRow(1);
     for (i = startItem; i < endItem; i++) {
          newrow = movieTable.insertRow(-1);
            newrow.insertCell(-1).innerHTML =
data[i].getElementsByTagName('rank')[0].text-
Content;
            newrow.insertCell(-1).innerHTML =
data[i].getElementsByTagName('name')[0].text-
Content;
              newrow.insertCell(-1).innerHTML
=   data[i].getElementsByTagName('earning')[0].
```

```
textContent;
        }
    }
    function gotoPage(direction){
        newpage = currentPage + direction;
        if(newpage < noOfPages && newpage >= 0)
                displayPage(newpage);
    }
```

In the updated 'displayPage' function, we now check if the page requested is '-1' or the last page, in which case, we go to the last page (the last page will be one less than the noOfPages, as the no of pages counts the 0th page as well). It also updates the global 'currentPage' variable with the current page number. Not that the variable 'currentPage' has to be declared once outside all the functions to be accessible globally.

In the new 'gotoPage' function, we first calculate the new page number, by adding 'direction' to the current page (`newpage = currentPage + direction`). Then we check if the new page number is within bounds, or basically that it is less than the total number of pages (`newpage < noOfPages`) and not less than zero (`newpage >= 0`). If the new page number is within bounds, we go directly to it (`displayPage(newpage)`).

Now take a look at the HTML code for the new navigation mechanism.

```
<table border="1">
        <tr>
      <td onclick="displayPage(0)">first</td>
     <td onclick="gotoPage(-1)">previous</td>
         <td onclick="gotoPage(1)">next</td>
         <td onclick="displayPage(-1)">last</
td>
        </tr>
     </table>
```

This can be placed between the data list, and the pages list. Before we take a look at the code as a whole, let us first add another feature quite popular in web sites. That is to let the user select how many items to show on the page. This can be done by giving the user a combo box / dropdown box containing a number of options. For our purposes, we will give the

user an option to choose between displaying 5, 10, 15, 20, 25, 30, 35, 40, 45 and 50. In a real application, we may limit these choices, but for now we consider as many as we can so that we can test how well the app works in different situations.

Rank	Name	Earning
31	GONE WITH THE WIND (1939)	198655278
32	Indiana Jones and the Last Crusade	197171806
33	Toy Story	191773049
34	Gladiator	186610052
35	SNOW WHITE & THE SEVEN DWARFS (1937)	184925486
36	DANCES WITH WOLVES (1990)	184208848
37	Batman Forever	184031112
38	Fugitive, The	183875760
39	Grease	181700000
40	Liar Liar	181410615

first previous next last

1 2 3 4 5 6 7 8 9 10

More flexible navigation

So how do we go about doing this? All we need to do is to create a dropdown box listing the no. of items to display per page and refresh the page every time this value is changed. Simple!

The reason it is that simple is because all our functions have been written taking into account the variable 'itemsPerPage'. Now it is merely a matter of changing the value of 'itemsPerPage' and then refreshing the contents of the page and the functionality is ready!

First we add a select box to out HTML page.

```
<p>
    No. of items to display per page:
<select id="itemsPerPage" onchange="changeItemsPerPage()">
</select>
</p>
```

This is added above the data list.

Now we create a function to generate options for this select box. You can of course have all of this statically present in the HTML file, as this will not change once the page loads. However, it will be much more controllable via JavaScript.

```
function generatePageOptions(from, to, step){
    ippSelect = document.getElementById('itemsPerPage');
    for (i = from; i <= to; i += step) {
        var newElem = document.
```

```
createElement("option");
            newElem.text = i;
            newElem.value = i;
            ippSelect.options.add(newElem);
        }
    }
```

This function is taking three parameters.

- The parameter 'from' tells the function the minimum value option for 'itemsPerPage' from where to start generating.
- The parameter 'to' tells the function the maximum value option for 'itemsPerPage' until which to generate options.
- The parameter 'step' defines how much fine grained control the user should have between values.

In our case, we want to generate options at a step of '5' between '5' and '50'. To do that we will add a call to this function in out 'onDataLoad' function. This way we have full extensibility, we can just as easily give options from '10' to '25' with a step of '3' between each.

One of the reasons we choose to have the odd option of 15, 35, 45 items per page is so that we encounter the case where the last page has less items than defined in 'itemsPerPage'. Here we will need to update our 'displayPage' function, to check for such a situation, and end the loop right there, instead of adding empty rows.

```
function displayPage(pagenum){
    if (pagenum == -1)
        pagenum += noOfPages;
    currentPage = pagenum;
    startItem = pagenum * itemsPerPage;
    endItem = (startItem + itemsPerPage);
  if(endItem > dataLength)
        endItem =  dataLength;

    while (movieTable.rows.length > 1)
        movieTable.deleteRow(1);

    for (i = startItem; i < endItem; i++) {
        newrow = movieTable.insertRow(-1);
          newrow.insertCell(-1).innerHTML =
```

```
data[i].getElementsByTagName('rank')[0].text-
Content;
                newrow.insertCell(-1).innerHTML =
data[i].getElementsByTagName('name')[0].text-
Content;
                  newrow.insertCell(-1).innerHTML
=   data[i].getElementsByTagName('earning')[0].
textContent;
        }
    }
```

As you can see all we added was a check for the situation when the calculated end item is out of the range of the data (`endItem > dataLength`), and in that case, terminate the data iteration at the last data item.

Finally we will need to update the contents of the page whenever the 'itemsPerPage' count changes. For this purpose we create a function called 'changeItemsPerPage' which will update the value, and calculate the new no. of pages, and then refresh the navigation menu and the data.

```
    function changeItemsPerPage(){
        itemsPerPage = Number(document.getElemen
tById('itemsPerPage').value);
        noOfPages = Math.ceil(dataLength / item-
sPerPage);
        displayPage(0);
        buildNavMenu();
    }
```

Now that we have implemented this functionality, let us have another look of what our code has become. Once again the JavaScript code and the HTML code will be given separate, but you can keep them in the same file.

```
    var dataLoader;
    var data;
    var dataLength = 0;
    var itemsPerPage = 10;
    var noOfPages = 0;
    var movieTable;
    var currentPage = 0;
```

```
   function createAJAXObject(){
        var httpReq;
        try {
            httpReq = new XMLHttpRequest();
        }
        catch (e) {
            try {
             httpReq = new ActiveXObject("Msxml2.
XMLHTTP");
            }
            catch (e) {
                try {
                                    httpReq = new
ActiveXObject("Microsoft.XMLHTTP");
                }
                catch (e) {
                    return false;
                }
            }
        }
        return httpReq;
   }
   function loadData(){
        dataLoader = createAJAXObject();
         dataLoader.onreadystatechange = onData-
Load;
           dataLoader.open('GET', 'movies.xml',
true);
        dataLoader.send(null);
   }
   function onDataLoad(){
        if (dataLoader.readyState != 4)
            return;
         if (dataLoader.status != 200 && dataLo-
ader.status != 0)
            return;
          data = dataLoader.responseXML.documen-
tElement.getElementsByTagName('movie');
         movieTable = document.getElementById('m
ovies');
```

```
        dataLength = data.length;
        noOfPages = data.length / itemsPerPage;
        generatePageOptions(5, 50, 5);
        changeItemsPerPage();
    }

    function generatePageOptions(from, to,
step){
        ippSelect = document.getElementById('ite
msPerPage');
        for (i = from; i <= to; i += step) {
                    var newElem = document.
createElement("option");
            newElem.text = i;
            newElem.value = i;
            ippSelect.options.add(newElem);
        }
    }

    function displayPage(pagenum){
        if (pagenum == -1)
            pagenum += noOfPages;
        currentPage = pagenum;
        startItem = pagenum * itemsPerPage;
        endItem = (startItem + itemsPerPage);

        if(endItem > dataLength)
            endItem = dataLength;

        while (movieTable.rows.length > 1)
            movieTable.deleteRow(1);

        for (i = startItem; i < endItem; i++) {
            newrow = movieTable.insertRow(-1);
              newrow.insertCell(-1).innerHTML =
data[i].getElementsByTagName('rank')[0].text-
Content;
              newrow.insertCell(-1).innerHTML =
data[i].getElementsByTagName('name')[0].text-
Content;
```

```
                newrow.insertCell(-1).innerHTML
=   data[i].getElementsByTagName('earning')[0].
textContent;
        }
   }
   function gotoPage(direction){
        newpage = currentPage + direction;
         if (newpage < noOfPages && newpage >=
0)
            displayPage(newpage);
   }

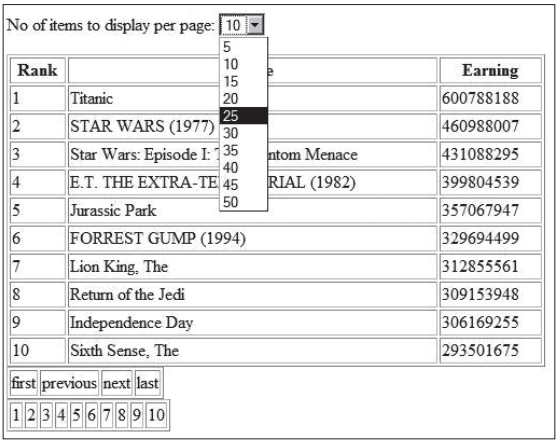
   function changeItemsPerPage(){
        itemsPerPage = Number(document.getElemen
tById('itemsPerPage').value);
        noOfPages = Math.ceil(dataLength / item-
sPerPage);
        displayPage(0);
        buildNavMenu();
   }
   function buildNavMenu(){
        navMenu = document.getElementById('navMe
nu').rows[0];
        navMenu.innerHTML = '';
        for (i = 0; i < noOfPages; i++) {
            newcell = navMenu.insertCell(-1);
            newcell.innerHTML = i + 1;
                    newcell.onclick   =   new
Function("displayPage(" + i + ");");
        }
   }

   <html>
     <head>
       <title>My AJAX App</title>
   <script  type="text/javascript"  src="script.
js">
   </script>
     </head>
     <body onload="loadData()">
```

```
    <p>
        No of items to display per page:
    <select id="itemsPerPage" onchange="changeIt
emsPerPage()">
    </select>
    </p>
                <table width="500" border="1"
id="movies">
          <tr>
            <th scope="col">Rank</th>
            <th scope="col">Name</th>
            <th scope="col">Earning</th>
          </tr>
        </table>
        <table border="1">
          <tr>
            <td onclick="displayPage(0)">first</
td>
               <td onclick="gotoPage(-1)">previ-
ous</td>
            <td onclick="gotoPage(1)">next</td>
            <td onclick="displayPage(-1)">last</
td>
          </tr>
        </table>
        <table id="navMenu" border="1">
          <tr>
            <td>
            </td>
          </tr>
        </table>
      </body>
    </html>
```

Now we have quite a powerful system for navigating a large database.

The app looks rather crude, so let's put in some effort to make it look slightly better. We will use a CSS style sheet, and include it in the document. Some more minor changes after that, and we have a much better looking app. You can get this



Now Navigation, and control over no. of items displayed! You're on a Roll!

style sheet along with all the example code in this book from the ThinkDigit web site. The style sheet and the HTML code including it are printed here in any case.

```
.navmenu td{
    border: thin solid #333;
    background-color: #666;
    color: #FFF;
    cursor: default;
}
.navmenu td:hover{
    border: thin solid #333;
    background-color: #333;
    color: #FFF;
    cursor: default;
}
#movies th {
    color: #000;
    background-color: #CCC;
    margin-right: 5px;
    margin-left: 5px;
    padding-right: 5px;
    padding-left: 5px;
    border-top-style: solid;
```

```
        border-top-width: thin;
        border-top-color: #000;
        border-bottom-color: #000;
        border-right-style: solid;
        border-left-style: solid;
        border-right-color: #FFF;
        border-left-color: #FFF;
        border-right-width: thin;
        border-left-width: thin;
    }
    #movies td {
        color: #000;
        padding-top: 0px;
        padding-bottom: 0px;
        margin-top: 0px;
        margin-bottom: 0px;
        background-color: #ddd;
        border-top-width: thin;
        border-bottom-width: thin;
        border-top-style: solid;
        border-bottom-style: solid;
        border-top-color: #999;
        border-bottom-color: #999;
    }
```

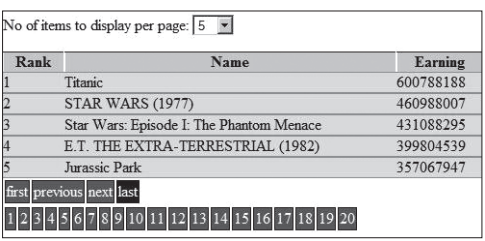
Place this code in a file called 'style.css' in the same folder as the other files.

Since we've already discussed CSS, we will not go into what is happening here.

```
    <html>
      <head>
        <title>My AJAX App</title>
      <script type="text/javascript" src="script.
js"></script>
        <link href="style.css" rel="stylesheet"
type="text/css" />
      </head>
      <body onload="loadData()">
          <p>No of items to display per page:
<select id="itemsPerPage"
        onchange="changeItemsPerPage()">
```

```
        </select></p>
         <table width="500" border="0" cellpad-
ding="0" cellspacing="0"
        id="movies">
          <tr>
            <th scope="col">Rank</th>
            <th scope="col">Name</th>
            <th scope="col">Earning</th>
          </tr>
        </table>
        <table class="navmenu">
          <tr>
            <td onclick="displayPage(0)">first</
td>
              <td onclick="gotoPage(-1)">previ-
ous</td>
            <td onclick="gotoPage(1)">next</td>
            <td onclick="displayPage(-1)">last</
td>
          </tr>
        </table>
        <table class="navmenu" id="navMenu">
          <tr>
            <td>
            </td>
          </tr>
        </table>
      </body>
    </html>
```

This is the HTML code with minor changes to better accommodate the CSS styles.



Just adding some attitude!

In our final act, we will now add functionality similar to Google Suggest in our app. For those

unfamiliar with Google Suggest, it is a search feature, where functionality such as auto-complete, and dropdown suggestions appear when you start typing your search terms.

Since this is a small offline database, we will simply have a small dropdown displayed below a search field which will contain possible matches and data about them.

First, let's add a search field right at the top, above everything else.

```
<p>
Quick look into the database:
<input id="suggestfield" type="text" onkeyup=
"getSuggestions()" onBlur="hideSuggestions()"/>
</p>
```

This text field will call the 'getSuggestions' function every time you enter a letter into the textbox (onkeyup), and when the textbox loses focus (onblur), the suggestions are hidden.

The suggestions themselves have to be displayed in some place. We will use a tag with a table in it for showing the suggestions.

```
<div id="suggestionsPane">
    Suggestions:<hr/>
    <table width="500" border="0" cellpad-
ding="0" cellspacing="0" id="suggestions">
        <tr>
        </tr>
    </table>
</div>
```

This is the content of the dropdown that will be displayed when the user starts typing in the 'suggestfield'. We rely on CSS for hiding and showing this element, so let us see the styles too.

```
#suggestionsPane {
    color: #000;
    background-color: #FFF;
    display: none;
    width: 500px;
    position: absolute;
    border-top-width: medium;
    border-right-width: medium;
    border-bottom-width: medium;
```

```
        border-left-width: medium;
        border-top-style: solid;
        border-right-style: solid;
        border-bottom-style: solid;
        border-left-style: solid;
        border-top-color: #CCC;
        border-right-color: #333;
        border-bottom-color: #333;
        border-left-color: #CCC;
        cursor: default;
        top: 25px;
    }
    #suggestionsPane tr{
        border-top-width: thin;
        border-right-width: thin;
        border-bottom-width: thin;
        border-left-width: thin;
        border-top-style: solid;
        border-right-style: none;
        border-bottom-style: solid;
        border-left-style: none;
        border-top-color: #666;
        border-right-color: #666;
        border-bottom-color: #666;
        border-left-color: #666;
    }
    #suggestionsPane tr:hover{
        background-color: #888;
    }
```

This might seem a lot for the job. However, most of it is just to make the 'suggestionsPane' look and behave more like a dropdown. The minimal CSS style sheet for this job would simply be:

```
    #suggestionsPane {
        display: none;
    }
```

Small isn't it! Basically, the only thing we want is that the suggestions be hidden till needed. After that, the rest is to make it behave like an element that has popped up, make it look like a list instead of a table, and to beautify it, etc. Add

the CSS code to the old file to get a better dropdown.

Now for the core functionality, we add the functions 'getSuggestions' and 'hideSuggestions'.

Lets look at 'getSuggestions' line by line:

- `searchtext = document.getElementById('suggest field').value;`
  This will just get the value entered in the suggestions field, and store it in the variable 'searchtext'.

- `document.getElementById('suggestionsPane'). style.display = 'block';`
  This will 'unhide' the 'suggestionsPane' by setting its display property from 'none' to 'block'.

- `document.getElementById('suggestions'). innerHTML = '';`
  This will clear the 'suggestionsPane'.

- `for (i = 0; i < dataLength; i++)`
  We now iterate over all the data.

- `if (data[i].textContent.indexOf(searchtext) > -1) {`
  This checks if the current row of data contains the text in the 'searchtext' variable.

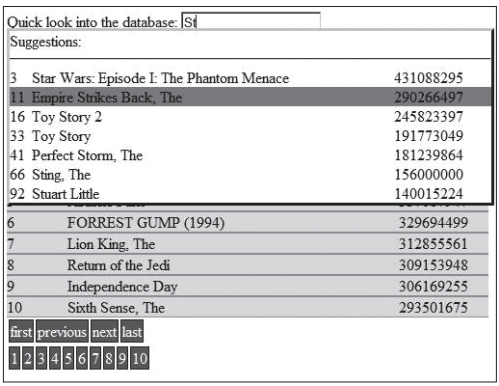
- ```
  newrow = document.getElementById('suggestions ').insertRow(-1);
  newrow.insertCell(-1).innerHTML = data[i]. getElementsByTagName('rank')[0].textContent;
  newrow.insertCell(-1).innerHTML = data[i]. getElementsByTagName('name')[0].textContent;
  newrow.insertCell(-1).innerHTML = data[i].get ElementsByTagName('earning')[0].textContent;
  ```
  These may seem familiar to you. This is the same procedure we used to add rows to the main data table 'movies' earlier, in the 'displayPage' function.

- `if (document.getElementById('suggestions'). rows.length > noOfSuggestions)`
  ```

```
break;
}
```

This code will stop the search once enough matches are found. You can control how many suggestions are shown in the dropdown using the 'noOfSuggestions' variable. In our code, we are setting it to 6.

The 'hideSuggestions' function is simple enough. All it does is sets the display style of the 'suggestionsPane' to 'none' so it is no longer displayed on screen.



AJAX suggestions

And that's it! We have a functional suggestions application!

Another small feature you can add to this, which will make it even more worthwhile is, to actually make it do something with the movies. How about we search in IMDB (The Internet Movie Database) for the movie whenever anyone double-clicks on a row containing the name of the movie? It is simple enough, we just need to update our 'displayPage' function to incorporate this functionality.

```
function displayPage(pagenum){
    if (pagenum == -1)
        pagenum += noOfPages;
    currentPage = pagenum;
    startItem = pagenum * itemsPerPage;
    endItem = (startItem + itemsPerPage);
    if (endItem > dataLength)
        endItem = dataLength;
    while (movieTable.rows.length > 1)
        movieTable.deleteRow(1);
    for (i = startItem; i < endItem; i++) {
```

```
            newrow = movieTable.insertRow(-1);
              newrow.insertCell(-1).innerHTML =
data[i].getElementsByTagName('rank')[0].text-
Content;
              newrow.insertCell(-1).innerHTML =
data[i].getElementsByTagName('name')[0].text-
Content;
                newrow.insertCell(-1).innerHTML
=   data[i].getElementsByTagName('earning')[0].
textContent;
             imdbsearch = "http://www.imdb.com/
find?s=tt&q=" +
   unescape(data[i].getElementsByTagName('name'
)[0].textContent);
        newrow.ondblclick = new Function('window.
location = "' + imdbsearch + '" ;');
       }
   }
```

We only needed to add two lines to get this functionality. Firstly, we generate a URL which will take us to the search page. This URL is (**http://www.imdb.com/find?s=tt&q=**) followed by the movie name. We use the 'unescape' function to make sure that the name of the movie is compatible with URL naming schemes.

The second line attaches a function to the 'ondblclick' (on double click) event of the row, which takes the user to the generated URL.

Now, in a final stroke of genius, we will add an awesome new feature to the 'suggestions' feature of the application. Search Highlighting.

Basically it will highlight the text we were searching for in the results. It sounds complicated and difficult, but really it isn't. Not that much!

For this we will modify the 'getSuggestions' function.

```
function getSuggestions(){
     searchtext = document.getElementById('su
ggestfield').value;
      document.getElementById('suggestionsPane
').style.display = 'block';
       document.getElementById('suggestions').
innerHTML = '';
```

```
        for (i = 0; i < dataLength; i++) {
                        if (data[i].textContent.
indexOf(searchtext) > -1) {
                    newrow = document.getElementByI
d('suggestions').insertRow(-1);

                                newrow.insertCell(-
1).innerHTML = data[i].getElementsByTagName('r
ank')[0].textContent.replace(searchtext, "<span
class='match'>" + searchtext + "</span>");

                                newrow.insertCell(-
1).innerHTML = data[i].getElementsByTagName('n
ame')[0].textContent.replace(searchtext, "<span
class='match'>" + searchtext + "</span>");

                    newrow.insertCell(-1).innerHT-
ML = data[i].getElementsByTagName('earning')
[0].textContent.replace(searchtext, "<span
class='match'>" + searchtext + "</span>");
            }
            if (document.getElementById('suggest
ions').rows.length > noOfSuggestions)
```

Now this looks REALLY complicated! But let's look only at the changes in the code. That would be the right-hand side of the three (`newrow.insertCell(-1).innerHTML =`) lines. Let's see what one of them does, and the others follow the same principle.

Instead of the original:

```
newrow.insertCell(-1).innerHTML = data[i].
getElementsByTagName('name')[0].textContent;
```

We now have:

```
    newrow.insertCell(-1).innerHTML =
data[i].getElementsByTagName('name')[0].
textContent.replace(
                    searchtext,
                    "<span class='match'>"
                    + searchtext +
                    "</span>"
                    );
```

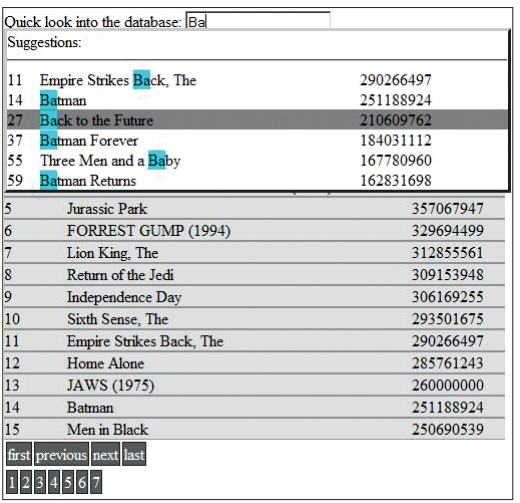
Let's break this down further:

Our original line was (`data[i].getElementsByTagName('name')[0].textContent`) which was retrieving the text content of the movie's name tag.

Now we are taking that text content, and replacing every instance of the content of the 'searchtext' variable (which is the value entered in the search box), and replacing it with (`"<span class='match'>" + searchtext + "</span>"`). This is basically enclosing the instances of the search text within a tag which has been assigned a style class of 'match'.

```
.match
{
    background-color: #00ffff;
}
```

Whatever style we now define in the CSS style for a class of 'match', will be applied to the instances of the search text! In our case we are highlighting it with a cyan background, but you can do whatever you want.



AJAX suggestions with search terms highlighting

Now our journey is complete. Let us look back once more at our creation in its full glory. Script, Style and HTML.

Script: script.js

```
var dataLoader;
var data;
var dataLength = 0;
var itemsPerPage = 10;
var noOfPages = 0;
var movieTable;
var currentPage = 0;
var noOfSuggestions = 6;
function createAJAXObject(){
     var httpReq;
     try {
          httpReq = new XMLHttpRequest();
     }
     catch (e) {
          try {
           httpReq = new ActiveXObject("Msxml2.
XMLHTTP");
          }
          catch (e) {
               try {
                                   httpReq = new
ActiveXObject("Microsoft.XMLHTTP");
               }
               catch (e) {
                    return false;
               }
          }
     }
     return httpReq;
}

function loadData(){
     dataLoader = createAJAXObject();
      dataLoader.onreadystatechange = onData-
Load;
        dataLoader.open('GET', 'movies.xml',
true);
     dataLoader.send(null);
}
```

```
   function onDataLoad(){
        if (dataLoader.readyState != 4)
            return;
        if (dataLoader.status != 200 && dataLo-
ader.status != 0)
            return;
         data = dataLoader.responseXML.documen-
tElement.getElementsByTagName('movie');
        movieTable = document.getElementById('m
ovies');
        dataLength = data.length;
        noOfPages = data.length / itemsPerPage;
        generatePageOptions(5, 50, 5);
        changeItemsPerPage();
   }

   function generatePageOptions(from, to,
step){
        ippSelect = document.getElementById('ite
msPerPage');
        for (i = from; i <= to; i += step) {
                    var newElem = document.
createElement("option");
            newElem.text = i;
            newElem.value = i;
            ippSelect.options.add(newElem);
        }
   }

   function gotoPage(direction){
        newpage = currentPage + direction;
       if (newpage < noOfPages && newpage >= 0)
            displayPage(newpage);
   }
   function changeItemsPerPage(){
        itemsPerPage = Number(document.getElemen
tById('itemsPerPage').value);
        noOfPages = Math.ceil(dataLength / item-
sPerPage);
        displayPage(0);
```

```
        buildNavMenu();
    }

    function displayPage(pagenum){
        if (pagenum == -1)
            pagenum += noOfPages;
        currentPage = pagenum;
        startItem = pagenum * itemsPerPage;
        endItem = (startItem + itemsPerPage);
        if (endItem > dataLength)
            endItem = dataLength;

        while (movieTable.rows.length > 1)
            movieTable.deleteRow(1);

        for (i = startItem; i < endItem; i++) {
            newrow = movieTable.insertRow(-1);
               newrow.insertCell(-1).innerHTML =
data[i].getElementsByTagName('rank')[0].text-
Content;
               newrow.insertCell(-1).innerHTML =
data[i].getElementsByTagName('name')[0].text-
Content;
                  newrow.insertCell(-1).innerHTML
=   data[i].getElementsByTagName('earning')[0].
textContent;
             imdbsearch = "http://www.imdb.com/
find?s=tt&q=" + unescape(data[i].getElementsByT
agName('name')[0].textContent);
         newrow.ondblclick = new Function('window.
location = "' + imdbsearch + '" ;');
        }
    }

    function buildNavMenu(){
        navMenu = document.getElementById('navMe
nu').rows[0];
        navMenu.innerHTML = '';
        for (i = 0; i < noOfPages; i++) {
            newcell = navMenu.insertCell(-1);
```

```
            newcell.innerHTML = i + 1;
                        newcell.onclick = new
Function("displayPage(" + i + ");");
        }
    }
    function getSuggestions(){
        searchtext = document.getElementById('su
ggestfield').value;
        document.getElementById('suggestionsPane
').style.display = 'block';
         document.getElementById('suggestions').
innerHTML = '';
        for (i = 0; i < dataLength; i++) {
            a = new String();
                    a.replace(searchtext, "<span
class='match'>" + searchtext + "</span>")
                        if (data[i].textContent.
indexOf(searchtext) > -1) {
                newrow = document.getElementByI
d('suggestions').insertRow(-1);

                                newrow.insertCell(-
1).innerHTML = data[i].getElementsByTagName('r
ank')[0].textContent.replace(searchtext, "<span
class='match'>" + searchtext + "</span>");

                                newrow.insertCell(-
1).innerHTML = data[i].getElementsByTagName('n
ame')[0].textContent.replace(searchtext, "<span
class='match'>" + searchtext + "</span>");

                    newrow.insertCell(-1).innerHT-
ML = data[i].getElementsByTagName('earning')
[0].textContent.replace(searchtext, "<span
class='match'>" + searchtext + "</span>");

                    imdbsearch = "http://www.imdb.
com/find?s=all&q=" + unescape(data[i].getElemen
tsByTagName('name')[0].textContent);
            newrow.onclick = new Function('window.
```

```
location = "' + imdbsearch + '" ;');
            }
            if (document.getElementById('suggest
ions').rows.length > noOfSuggestions)
                break;
        }
    }
```

HTML: index.html

```
    <html>
      <head>
        <title>My AJAX App</title>
    <script  type="text/javascript"  src="script.
js">
    </script>
         <link href="style.css" rel="stylesheet"
type="text/css" />
      </head>
      <body onload="loadData()">
         <p>Quick look into the database: <input
id="suggestfield"
        type="text" onkeyup="getSuggestions()"
        onblur="hideSuggestions()" /></p>
                  <div    id="suggestionsPane"
class="suggestionsPane">
          Suggestions:
          <hr />
          <table width="500" border="0" cellpad-
ding="0" cellspacing="0"
          id="suggestions">
            <tr>
              <td>
              </td>
            </tr>
          </table>
        </div>
          <p>No  of  items  to  display  per  page:
<select id="itemsPerPage"
        onchange="changeItemsPerPage()">
```

```
        </select></p>
         <table width="500" border="0" cellpad-
ding="0" cellspacing="0"
        id="movies">
          <tr>

            <th scope="col">Rank</th>
            <th scope="col">Name</th>
            <th scope="col">Earning</th>
          </tr>
        </table>

        <table class="navmenu">
          <tr>
         <td onclick="displayPage(0)">first</td>
        <td onclick="gotoPage(-1)">previous</td>
            <td onclick="gotoPage(1)">next</td>
         <td onclick="displayPage(-1)">last</td>
          </tr>

        </table>
        <table class="navmenu" id="navMenu">
          <tr>
            <td>
            </td>
          </tr>
        </table>
      </body>
    </html>
```

Style: style.css

```
.navmenu td{
    border: thin solid #333;
    background-color: #666;
    color: #FFF;
    cursor: default;
}

.navmenu td:hover{
```

```
        border: thin solid #333;
        background-color: #333;
        color: #FFF;
        cursor: default;
    }

    #movies th {
        color: #000;
        background-color: #CCC;
        margin-right: 5px;
        margin-left: 5px;
        padding-right: 5px;
        padding-left: 5px;
        border-top-style: solid;
        border-top-width: thin;
        border-top-color: #000;
        border-bottom-color: #000;
        border-right-style: solid;
        border-left-style: solid;
        border-right-color: #FFF;
        border-left-color: #FFF;
        border-right-width: thin;
        border-left-width: thin;
    }

    #movies td {
        color: #000;
        padding-top: 0px;
        padding-bottom: 0px;
        margin-top: 0px;
        margin-bottom: 0px;
        background-color: #ddd;
        border-top-width: thin;
        border-bottom-width: thin;
        border-top-style: solid;
        border-bottom-style: solid;
        border-top-color: #999;
        border-bottom-color: #999;
    }
```

```
.match
{
    background-color: #00ffff;
}

#suggestionsPane {
    color: #000;
    background-color: #FFF;
    display: none;
    width: 500px;
    position: absolute;
    border-top-width: medium;
    border-right-width: medium;
    border-bottom-width: medium;
    border-left-width: medium;
    border-top-style: solid;
    border-right-style: solid;
    border-bottom-style: solid;
    border-left-style: solid;
    border-top-color: #CCC;
    border-right-color: #333;
    border-bottom-color: #333;
    border-left-color: #CCC;
    cursor: default;
    top: 25px;
}

#suggestionsPane tr{
    border-top-width: thin;
    border-right-width: thin;
    border-bottom-width: thin;
    border-left-width: thin;
    border-top-style: solid;
    border-right-style: none;
    border-bottom-style: solid;
    border-left-style: none;
    border-top-color: #666;
    border-right-color: #666;
    border-bottom-color: #666;
    border-left-color: #666;
```

```
}
#suggestionsPane tr:hover{
    background-color: #888;
}
```

5.4 Conclusion

Now that you have developed in AJAX yourself, you understand what all the fuss is about. With a simple and free text editor, you can create powerful web applications that touch the bounds of interactivity. Never be shy in exploring how a site works. Go ahead and press `[Ctrl]` + `[U]` to see the code in all its glory. Download all the attached scripts you find on web sites and see what they're doing, and how.

You can use the FireBug add-on for Firefox to see live code in action, make changes on your code on the fly, and see how it works out. The best thing about the web is that almost everything is in pure text, so you need nothing more than a browser and text editor to experiment.

Even if you don't plan to become a full-fledged web developer, the pure joy you get in coding something for the web is worth it. Entire communities are available online dedicated to helping out new developers. So go ahead and bring your unique perspective to the web.

Here we leave you off, hopefully with enough acceleration towards AJAX to break through as a web developer. However, this is not the end — keep playing and experimenting!

JavaScript Frameworks

6.1 Introduction

In the last section, we developed a simple and small navigation interface for a movie database that could:

- Display the data in pages
- Allow the user to select the size of pages
- Navigate to any page directly
- Navigate using back and forward buttons
- Show suggestions when you start typing a movie name
- Highlight the occurrences of your search terms in the suggestions

A lot of what we did is functionality that many applications share. Any application that has a lot of data will display it in pages and will probably give you control over how much data is displayed. Unless the number of pages is high, such applications will also show page numbers.

Auto-suggestion is also a very popular feature. You can see them when you start typing a friend's name in Gmail, or a search term in Google Suggest. Search highlighting is also prevalent in most sites that provide search facility. These are some very common things that people do with AJAX and JavaScript. As such, there are many ready-to-use code libraries available for performing repetitive tasks.

These libraries take the major job off your hands. You no

longer need to focus on bringing your web site up to par with standards — you need only to make it better.

6.2 A comparison of frameworks

Since different frameworks have completely different ways of doing the same tasks, it's impossible to learn them all here. Most frameworks aim to achieve the same results. So the difference between them, is how they can be used, their size, their speed, etc.

Every framework tries to give the developer a unique way of dealing with the same tasks, and thus learning each one is a different experience. While many try to do everything entirely in JavaScript, others allow you to use special markup in your page to accomplish powerful feats. Going into details about all frameworks is not possible, as each could take up a book its own right, instead we will give an overview of the different frameworks available, and the facilities they provide:

- **Dojo toolkit**

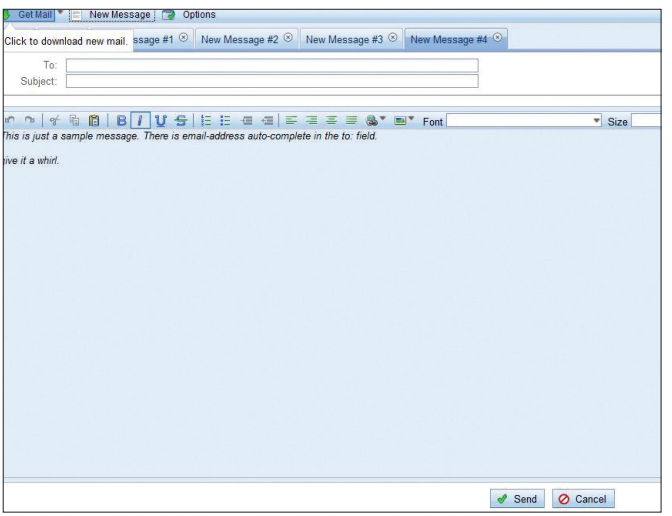
This is one of the most popular, free and open-licensed toolkits for JavaScript. It also has some features for backend / server-side integration with its code. This library is also well supported by major development platforms. It is integrated with newer versions for the Zend PHP platform. The features provided by this toolkit as listed by their web site are listed below.

It is available in three parts:

- **Core:** This provides basic JavaScript features such as
  - Browser detection
  - JSON encoding / decoding
  - Powerful AJAX support
  - Animation support
  - Asynchronous programming support
  - CSS style and positioning utilities
  - Object-oriented programming (OOP) support
  - Drag-and-drop

- Localisations
- Date, Number formatting
- String utilities
- Progressive-enhancement behaviour engine
- Cookie handling

- **dijit**: This is collection of widgets, i.e. individual components of HTML and JavaScript code that can be used anywhere. The widgets can easily be themes so that they blend in with your web site. It provides the following widgets:
  - Menus, tabs and tooltips
  - Sortable tables, dynamic charts and 2-D vector drawings
  - Tree widgets that support drag-and-drop
  - Various forms and routines for validating form input
  - Calendar-based date selector, time selector, and clock
- **dojoX**: This is a library for advanced data visualisation components such as charts and bar-graphs. It also has components for vector drawing, galleries, etc. It even provides facilities to make your application work better offline, using Google Gears.

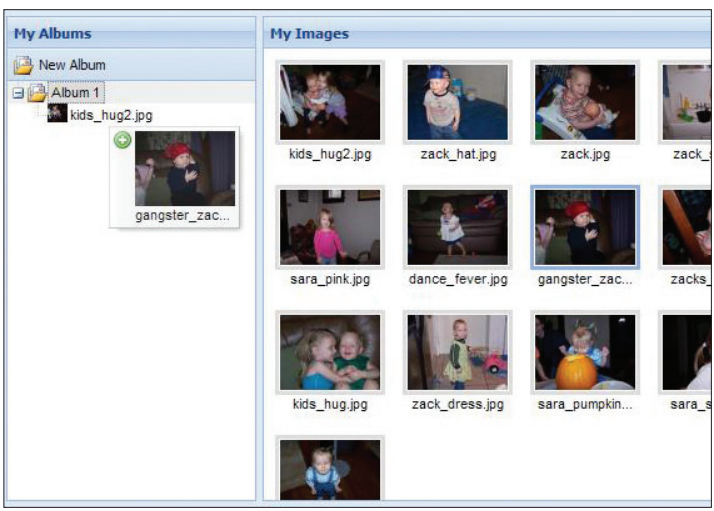


Ext

This is a powerful library for building rich GUIs. It uses AJAX in its core for data communication, synchronisation, etc. The latest version, according to their web site, provides widgets for:

- Text field and text area input controls
- Date fields with a pop-up date-picker
- Numeric fields
- List box and combo boxes
- Radio and checkbox controls
- HTML editor control
- Grid control
- Tree control
- Tab panels
- Toolbars
- Desktop-application-style menus
- Region panels to allow a form to be divided into multiple sub-sections
- Sliders

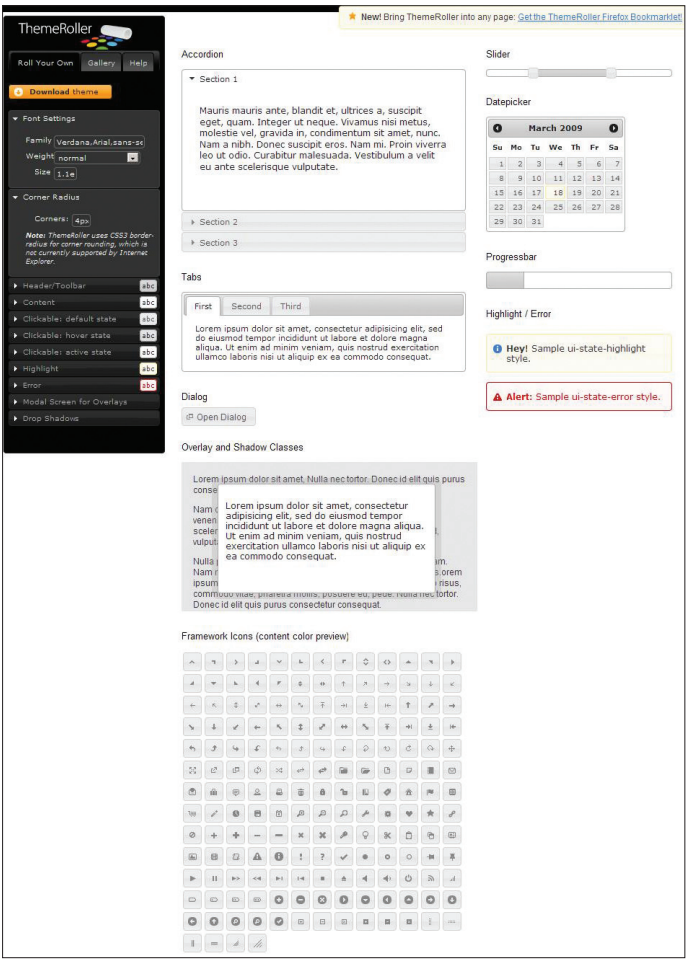
Ext JS



jQuery

A truly unique and very powerful library, and also one of the most popular ones, is jQuery. It is used even by organisations such as Google, IBM, Twitter and Amazon. Microsoft will incorporate this as a web development library for Visual Studio, and Nokia as a web runtime.

Its power lies in its very simplicity, as the most used and



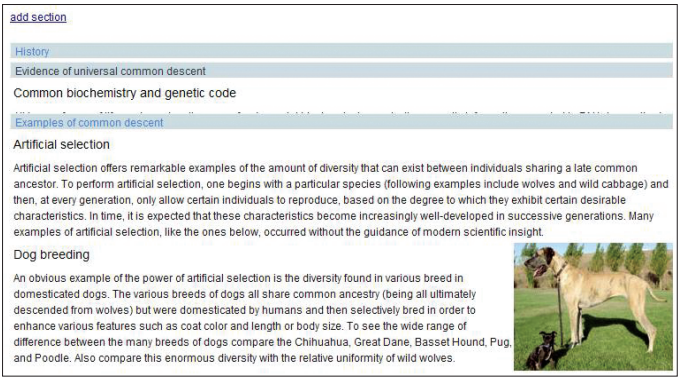
powerful component provides is a simple '$' function. It includes interactions for making elements draggable, droppable, resizable, selectable and sort able. It also provides some widgets for Accordion, Datepicker, Dialog, Progressbar, Slider and Tabs. It also supports theming and effects.

MooTools

Don't let the name mislead you! This is a very powerful, object-oriented framework, and is modular in its very essence. You need not include the entire library if you will only use a part. In fact, their web site allows you to choose what components you want to include in your library, and it will include only those in your download.

This kind of modularity is not unique to this library; however, it provides a much more fine grained control than most libraries.

The library itself is divided into two libraries: Core and More. The core library as its name suggests has basic features, while the More library has more advanced components. According to their web site their



feature set is as follows:

The Core library has the following modules:

- **Core:** Provides core features used by the rest of the library
- **Native:** Provides features to work with JavaScript native data types of Array, Function, Number, String, Hash and Event
- **Class:** Provides features to ease working with classes you create or extend. It has the modules Class and Class.Extras
- **Element:** Contains powerful modules to manipulating HTML elements. It contains the modules Element, Element.Event, Element.Style, Element.Dimensions
- **Utilities:** Some miscellaneous utilities for CSS, JSON coding, cookies, flash files, etc. it has the modules Selectors, DomReady, JSON, Cookie, Swiff
- **Fx:** Effects and transition libraries, composed of Fx, Fx.CSS, Fx.Tween, Fx.Morph, and Fx.Transitions
- **Request:** Provides features for easy working with Ajax using XML, HTML or JSON. Consists of Request, Request.HTML, Request.JSON

The More library consists of:

- **Fx:** More effects using Fx.Slide, Fx.Scroll, Fx.Elements
- **Drag:** Support for drag-and-drop operations using Drag, Drag.Move
- **Utilities:** More utilities, for handling cookies, colours, events and assets. Composed of the modules Hash. Cookie, Color, Group, Assets
- **Interface:** Interface elements and widgets such as Sortables, Tips, SmoothScroll, Slider, Scroller and Accordion

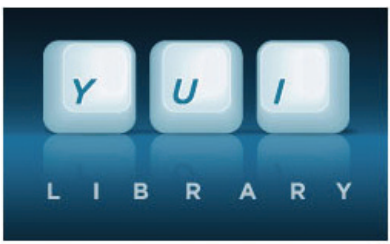
Prototype and script.aculo.us

Prototype is a JavaScript framework, and script.aculo.us is an add-on for the framework. Prototype's major strengths com-

pared to other frameworks are its features for creating classes, and extensions to DOM. script.aculo.us adds animation and effect features to Prototype.

The YahooUI library

This is a free open source library provided by Yahoo! for developers. Furthermore, it is under a non-restrictive licence, so you are free to modify and distribute it. It's very powerful, and supports many features. Similar to other libraries, it is also quite modularised.



uneditable	address	city	state	amount	active	colors	last_login
	1236 Some Street	San Francisco	CA	5	yes	red	04/19/2007
	3271 Another Ave	New York	NY	3	no	red,blue	02/15/2006
	9996 Random Road	Los Angeles	CA	0	maybe	green	01/23/2004
	1623 Some Street	San Francisco	CA	5	yes	red	04/19/2007
	3217 Another Ave	New York	NY	3	no	red,blue	02/15/2006
	9899 Random Road	Los Angeles	CA	0	maybe	green	01/23/2004
	1723 Some Street	San Francisco	A	5	yes	red	04/19/2007
	3241 Another Ave	New York	NY	3	no	red,blue	02/15/2006
	9909 Random Road	Los Angeles	CA	0	maybe	green	01/23/2004
	1623 Some Street	San Francisco	CA	5	yes	red	04/19/2007
	3721 Another Ave	New York	NY	3	no	red,blue	02/15/2006
	9989 Random Road	Los Angeles	CA	0	maybe	green	01/23/2004
	1293 Some Street	San Francisco	CA	5	yes	red	04/19/2007
	3621 Another Ave	New York	NY	3	no	red,blue	02/15/2006
	9959 Random Road	Los Angeles	CA	0	maybe	green	01/23/2004
	6123 Some Street	San Francisco	CA	5	yes	red	04/19/2007
	3281 Another Ave	New York	NY	3	no	red,blue	02/15/2006
	9989 Random Road	Los Angeles	CA	0	maybe	green	01/23/2004

The features as listed by the Yahoo UI JavaScript library are as follows:

- **YUI Core:** The YAHOO Global, DOM Collection, Event Utility
- **YUI Library Utilities:** Animation Utility, Browser History Manager, Connection Manager, Cookie Utility, DataSource Utility, Drag and Drop Utility, Element Utility, Get Utility, ImageLoader Utility, JSON Utility, Resize Utility, Selector Utility, StyleSheet, Utility beta, The YUI Loader Utility.
- **YUI Library Controls/Widgets:** AutoComplete, Button, Calendar, Carousel, Charts, Color Picker, Container, DataTable, ImageCropper, Layout Manager, Menu, Paginator, Rich Text Editor, Slider, TabView, TreeView, Uploader.
- **YUI Library CSS Tools:** CSS Reset, CSS Base, CSS Fonts, CSS Grids.

Spry

If you've used a recent version Adobe Dreamweaver, you will notice that it provides UI components and AJAX features using this library. Spry is a free and open source library developed by Adobe. It comes integrated with Dreamweaver CS3, and CS4.

Spry is more designer centric, and provides features that are implemented partly in the HTML code and partly in JavaScript. It too is very modularised, with modules such as Spry Effects, Spry Data and Spry Widgets.

Even if you are not using Dreamweaver you can download and use this library with any other web development environment. In the next section, we will use this library to make port the application we made in the last few chapters to Spry.

Deciding which library to use for your next application can be a daunting task, especially if you consider that once the choice is made you will have to stick to it for quite some time. The path you take while developing your application becomes dependant on the path taken by the library it implements. Features and flexibility are important; however, it is always better to go for a framework that is actively maintained, otherwise, you may be left with an application that has as many bugs as the library it uses.

In the next section, we take a look at how one would develop the movie database we made in the last section, using Spry.

6.3 Using Spry: Reworking our old example

Although it is impossible to touch all the different frameworks, looking at how you can develop using one is a good idea. We will now try to rebuild the movie database using Spry. Since this is meant only to be a peek, we will not discuss the features of Spry in detail, nor will we see how the Spry part of the code works. Instead, let us focus on how to get similar functionality using Spry.

If you're using Dreamweaver or Aptana you're all set, as Spry is included, otherwise, you can get it from the Digit DVD, or online from: **http://labs.adobe.com/technologies/spry/home.html**

This time around, we'll just jump straight to the code and see how Spry does things.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Transitional//EN"
       "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">
<html xmlns:spry="http://ns.adobe.com/spry/"
xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>My Ajax App</title>
<script src="lib/xpath.js" type="text/javas-
cript">

</script>
```

```
   <script   src="lib/SpryData.js"   type="text/
javascript">

   </script>
   <script src="lib/SpryPagedView.js" type="text/
javascript">

   </script>
   <script      src="lib/SpryAutoSuggest.js"
type="text/javascript">

   </script>
           <link  href="lib/SpryAutoSuggest.css"
rel="stylesheet"
       type="text/css" />
        <link href="style.css" rel="stylesheet"
type="text/css" />
   <script type="text/javascript">
         <!--
              var  moviesds  =  new  Spry.Data.
XMLDataSet("movies.xml", "movies/movie");
           moviesds.setColumnType("rank", "num-
ber");
        moviesds.setColumnType("earning", "num-
ber");

              var  suggests  =  new  Spry.Data.
XMLDataSet("movies.xml", "movies/movie");

            var  moviesPage  =  new  Spry.Data.
PagedView(moviesds, {
              pageSize: 5
         });

         function generatePageOptions(from, to,
step){
              ippSelect = document.getElementByI
d('itemsPerPage');
             for (i = from; i <= to; i += step)
{
```

```
                        var newElem = document.
createElement("option");
                    newElem.text = i;
                    newElem.value = i;
                                ippSelect.options.
add(newElem);
                }
            }

            //-->

    </script>
      </head>

       <body onload="generatePageOptions(5,50,5)
;">
        <div id="suggestionsPanel">
            <input id="suggestfield" type="text"
/>

          <div id="suggestions">
              <table border="0" cellpadding="0"
cellspacing="0"
             spry:region="suggests">
                <tr spry:repeat="suggests" spry:
suggest="{name}">
                 <td>{rank}</td>

                 <td>{name}</td>
               </tr>
             </table>
          </div>
        </div>

          <p>No of items to display per page:
<select id="itemsPerPage"
           onchange="moviesPage.setPageSize(this.
value)">
        </select></p>
    <table width="500" border="0" cellpadding="0"
```

```
cellspacing="0"
       spry:region="moviesPage">
         <tr>
           <th scope="col" onclick="moviesPage.
sort('rank')">Rank</th>

           <th scope="col" onclick="moviesPage.
sort('name')">Name</th>

           <th scope="col" onclick="moviesPage.
sort('earning')">Earning</th>
         </tr>

        <tr spry:repeat="moviesPage">
           <td>{rank}</td>

           <td>{name}</td>

           <td>{earning}</td>
         </tr>
   </table>

       <table class="navmenu">
         <tr>
                      <td  onclick="moviesPage.
firstPage()">first</td>

           <td onclick="moviesPage.previousPage
()">previous</td>

                      <td  onclick="moviesPage.
nextPage()">next</td>

                      <td  onclick="moviesPage.
lastPage()">last</td>
         </tr>
       </table>
   <script type="text/javascript">
         <!--
          var suggestWidget = new Spry.Widget.
```

```
AutoSuggest("suggestionsPanel",   "suggestions",
"suggests", "name", {
                hoverSuggestClass: 'hover',
                minCharsType: 0,
                containsString: false,
                maxListItems: 0
          });

          -->

   </script>
     </body>
   </html>
```

Spry is a huge and complicated JavaScript library, the entire code for Spry would probably fill up a book as large as this! So let's leave that out.

We'll start with the <head> tag. The things you'll notice are, the new scripts that have been added: xpath.js, SpryData.js, SpryPagedView.js and SpryAutoSuggest.js, also included is a new stylesheet SpryAutoSuggest.css.

A little about what each of these does:

- **xpath.js**
  XPath is a way of getting data from an XML file. It is similar a looking at an XML file like a folder and file structure. So if you have the following segment of XML code:

```
<movies>
      <movie>
          <rank>2</rank>
          <name>STAR WARS (1977)</name>
          <earning>460988007</earning>
      </movie>
</movies>
```

  The XPath to get "STAR WARS (1977)" would be "movies/movie/name". For a larger lists such as:
  You can retrieve a list of all movies by using "movies/movie".
  This file provides this functionality to the framework, but is not directly used in our app.

- **SpryData.js**
  This is a file for loading and manipulating data. It is the one responsible for performing the Asynchronous loading of our 'movies.xml' file, and sorting it.
- **SpryPagedView.js**
  To divide the data in pages as we do in our app, the features provided by Spry's PagedView come in handy. It allows us to divide the data in pages of any size we want, and navigate them with ease.
- **SpryAutoSuggest.js**
  This JavaScript file is a widget which provides AutoSuggest functionality like the suggestions we used in out app.

Note that only in features that lie in context of what we are trying to do here, have been listed. These files are capable and responsible for doing much more!

Moving down into the final script block we see the code segment:

You will recognise the 'generatePageOptions' function from our old example, and it is untouched. However, the rest of the code warrants an explanation.

- `var moviesds = new Spry.Data.XMLDataSet("movies.xml", "movies/movie");`
  This is the first time we are actually using Spry code. Here we use the 'XMLDataSet' feature provided by Spry to load the 'movies.xml' file, and retrieve the list of all movies from it. The first parameter is the name of the file to load, and the second is the XPath to the data that we want. We want a list of all the movies, and as such we use the XPath "movies/movie". This data set that we finally get is stored in the variable 'moviesds'.

- `moviesds.setColumnType("rank", "number");`
  `moviesds.setColumnType("earning", "number");`
  Here we tell the Spry how to treat the loaded data. We tell it that the 'rank ' and 'earning' columns are numbers. We can do without this if we want, but it is

better if we include it, as this allows better sorting of the data.
Numbers are sorted as 1,2,3,10,11,100; however the same if treated as text would sort as '1', '10', '11', '100', '2', '3'. Hence by telling Spry what the data is, we ensure that it is handled the right way.

- `var suggests = new Spry.Data.XMLDataSet("movies.xml", "movies/movie");`
This is another data set. This is to provide data to the AutoSuggest feature. As you can see, it is using the same data as 'moviesds'.

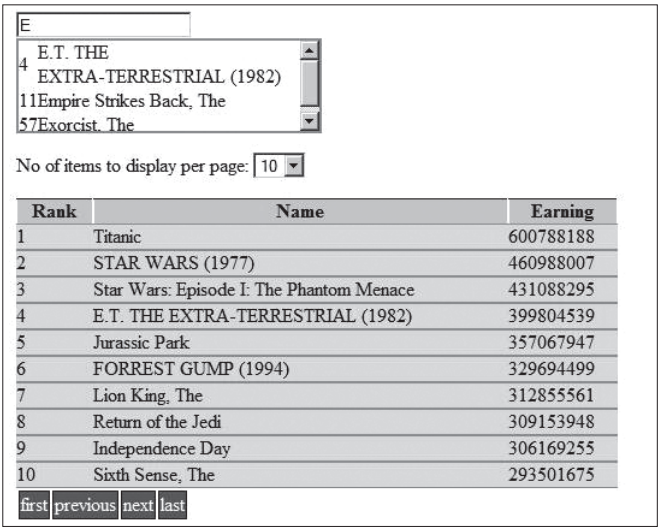
- `var moviesPage = new Spry.Data.PagedView(moviesds, { pageSize: 5 });`
The 'Spry.Data.PagedView' object enables us to break down the data that we are showing in the app, into pages. We provide it two parameters — the first is the name of the dataset that we wish to 'page', and the second are a list of options based on which the 'breaking'

No of items to display per page: 10

Rank	Name	Earning
1	Titanic	600788188
2	STAR WARS (1977)	460988007
3	Star Wars: Episode I: The Phantom Menace	431088295
4	E.T. THE EXTRA-TERRESTRIAL (1982)	399804539
5	Jurassic Park	357067947
6	FORREST GUMP (1994)	329694499
7	Lion King, The	312855561
8	Return of the Jedi	309153948
9	Independence Day	306169255
10	Sixth Sense, The	293501675

first previous next last

Our old app, now using spry



Spry Autosuggest at work

will be done. We are just setting one option, 'pageSize' which represents the no of items in each page. We are using a value of '5' like our original app.

While we are looking at the scripts, let us get another one over with, the one right at the bottom of the page.

```
<script type="text/javascript">
        <!--
         var suggestWidget = new Spry.Widget.
AutoSuggest("suggestionsPanel",  "suggestions",
"suggests", "name", {
              minCharsType: 0,
              containsString: false,
              maxListItems: 0
        });
        -->
</script>
```

This is actually a single line of code spread over multiple lines, to make it is easier to read. It is essentially telling Spry to treat the html element with the id 'suggestionsPanel' as the suggestions widget. The second parameter is the element

which will show the suggestions, in our case that would be the with the id "suggestions". The third parameter is the name of the dataset from which to get the results. The fourth parameter is the column of the dataset in which to search.

The final parameter is a list of options, which control the functioning of the feature. Here the options we set are:

- **minCharsType**: This controls how any characters need to be entered before the autosuggest starts looking for matches. Here we set it to 0, to make it always search.
- **containsString**: This setting allows us to optionally search the whole text, instead of just the first few characters.
- **maxListItems**: This controls how many items are displayed in the list. To list all items we set it to '0'.

That is all the code that is required in JavaScript! You must be thinking, "But we didn't do anything!" You're right Spry takes care of it all. All we need to do is where to put the data, and how to put it there.

How do we tell Spry where to put the data? Let's see.

```
<table width="500" border="0" cellpadding="0"
cellspacing="0"
       spry:region="moviesPage">
         <tr>
           <th scope="col" onclick="moviesPage.
sort('rank')">Rank</th>

           <th scope="col" onclick="moviesPage.
sort('name')">Name</th>

           <th scope="col" onclick="moviesPage.
sort('earning')">Earning</th>
         </tr>

        <tr spry:repeat="moviesPage">
           <td>{rank}</td>

           <td>{name}</td>
           <td>{earning}</td>
         </tr>
  </table>
```

The things to note here are the "spry:region" and "spry:repeat" attributes. These tell Spry how to hande this part of the page. The 'spry:region="moviesPage"' attribute tells Spry that we want the contents of this element to be processed by Spry before being outputted on the page. The 'spry:repeat="moviesPage"' attribute tells Spry to repeat the contents of this element for each item in the dataset 'moviesPage'.

Now all we do is put the data fields we want to output on the page within braces e.g. '{rank}'. Spry then replaces these with the actual values while processing the page.

Another amazing thing we have done in this app is something we didn't do in the original. We made the columns sortable! Now all you need to do is click on a column title, and it will sort the column!

How did we do this? Simply by adding an 'onclick' attribute to the cells of the header, telling Spry to sort the data. Take a look at the header tags we have now:

```
<th scope="col" onclick="moviesPage.sort('earning')">Earning</th>
```

The code "moviesPage.sort('earning')" is a JavaScript function that sorts the dataset by the column 'earning' and hence it is attached to the 'Earning' column heading. Similarly we add sort functions to the 'Rank' and 'Name' columns.

The navigation menu too is much simpler now. All the actions we need to perform are already supported by Spry.

```
    <table class="navmenu">
        <tr>
                    <td onclick="moviesPage.firstPage()">first</td>
          <td onclick="moviesPage.previousPage()">previous</td>
                    <td onclick="moviesPage.nextPage()">next</td>
                    <td onclick="moviesPage.lastPage()">last</td>
        </tr>
      </table>
```

This code needs no explanation since it's purpose and action is fairly evident.

One final thing that needs explanation is how we facilitate selecting the number of items to display per page.

```
<select id="itemsPerPage"
  onchange="moviesPage.setPageSize(this.value)">
</select>
```

This is all that is required to change the number of items to display per page. The select box starts off with the number of items, and the items are added in runtime the same way they were in our original app.

When the value is changed by selecting one from the dropdown, it fires off the code in the 'onchange' attribute, which is 'moviesPage.setPageSize(this.value)'. This code tells the Paging system of Spry to change the size of each page (i.e. the number of items displayed per page), to the value of the select field.

Here we conclude our experimentation with Spry. Keep in mind that this is just a small example of what Spry is capable of.

6.4 Conclusion

Now that you have learnt about frameworks and even used one yourself. Which one will you choose?

Many people make the mistake of going by the first advice they get. You've played around with Spry now, and you may feel like it suits you, or can do whatever you want to do. Don't go by what we say! Make up your own mind. Every framework is different, look them up and see what they have to offer. There are too many frameworks out there to be listed here, but we've tried to list a few of the common ones.

Each framework is designed by its developer to adhere to a different philosophy. Some of the simplest frameworks sometimes have the greatest potential and may surprise you. JQuery, for example, revolves around a simple function named '$'. However, the way it is designed to operate with that alone, makes it more powerful than perhaps some of the most complicated and extended libraries. Spry is a framework that even a designer can feel comfortable with. Finally, what you have to decide is which design philosophy resonates best with how you think and envision what you wish to accomplish.

Conclusion

7.1 Where to go from here

One thing you must have realised is that there is still a lot more you can learn about AJAX. We have only touched on one JavaScript Framework, and have used only basic AJAX. The thing with AJAX though is that all you can really learn from others are examples of how it can be used. It is merely a programming practice, and doesn't have a fixed syntax of its own.

If you're a web developer who has never heard of AJAX (although that is hard to imagine these days), and someone told you 'Look, here's a neat trick, how about you load the XML data asynchronously using JavaScript?', you could probably just jump with joy and start coding something immediately. AJAX is not a language, and thus all you really need is an existing background in JavaScript programming to start using it immediately, and that is what we've tried to develop here.

AJAX itself involves the usage of multiple technologies; however, combining it with more technologies can get you even further. For example, here are two technologies that can greatly improve a user's experience:

Adobe AIR:

Adobe AIR or Adobe Integrated Runtime, is a software development kit (SDK), which allows you bring your apps online. Using the AIR SDK, you can make installable applications with Flash and Flex using ActionScript, or using JavaScript and HTML. It is integrated well with Dreamweaver, and all you need to do is to export your app as an AIR file, and it's done!

The AIR SDK is available for free from Adobe's web site and it would do you well to check it out. It's truly encouraging to know that even if you've only done web development, you're not limited to the web alone, and all the work you put in on your site, can be reused to bring the same functionality offline.

Google Gears:

Google Gears gives developers another way of bringing an app offline. It is not a competitor for AIR, it is quite a different product and accomplishes different goals.

If you've used Google Docs, you will notice that they provide a facility to use the website even when offline, via a small link saying 'Offline' in the top right corner of the page. If you've never used it, here's what it does: it allows you to use Google Docs, even when you're disconnected from the internet. You'll still have all the same functionality, but you will be able to do everything you do in Google Docs, such as create a new document, or edit an existing document. These changes are stored locally on your computer, and when you go online, they are synchronised with the Google Docs server. Gmail also provides this facility now, however it needs to be turned on separately in Gmail Labs as it is still in development.

All Gears needs to function is a plug-in that is available for free in the Google Docs web site.

Many other technologies exist which leverage the power of AJAX, even for the facilities that the technologies above provide, many alternatives exist. All in all, learning how to harness the power of AJAX is a good idea even if you plan to

develop applications for the desktop, or what your application to be available offline.

So now that you can see how far AJAX reaches out, you should know where you can go to find out more. For this we have provided in the next chapter, a list of resources available for your perusal. So onward! You have places to go and things to do!

7.2 Web development software and tools

In this article, in many places we've mentioned different technologies and products, and here we list where they can be found on the web.

Web Development IDE

Name	Web site	License type(s)
Aptana	http://www.aptana.com/	Free and Commercial (Pro)
Adobe	http://www.adobe.com/products/dreamweaver/	Commercial
Zend Studio	http://www.zend.com/	Commercial
Microsoft Expression Studio	http://www.microsoft.com/expression/	Commercial and Free (Express)
Quanta Plus	http://quanta.kdewebdev.org/	Open Source
Eclipse with Web Tools Platform	http://www.eclipse.org/webtools/	Open Source

XML Editing Tools

Name	Web site	License type(s)
Altova XMLSpy	http://www.altova.com/	Commercial
oXygen XML editor	http://www.oxygenxml.com/	commercial
OpenXML editor	http://www.philo.de/xmledit/	
XMLmind	http://www.xmlmind.com/xmleditor/	Free (Personal) and Commercial (Professional)
XML Copy Editor	http://xml-copy-editor.sourceforge.net/	Open Source
Conglomerate	http://www.conglomerate.org	Open Source
Syntex Serna	http://www.syntext.com/	Free and Commercial (Serna Enterprise)
Xerlin	http://www.xerlin.org/	Open Source

WYSIWYG HTML Software

Name	Web site	License type(s)
Amaya	http://www.w3.org/Amaya/	Open Source
Komposer	http://kompozer.net/	Open Source
N|vu	http://www.net2.com/nvu/	Open Source
BlueGriffon	http://bluegriffon.org/	Open Source
Adobe Dreamweaver	http://www.adobe.com/products/dreamweaver/	Commercial
Zend Studio	http://www.zend.com/	Commercial
Microsoft Expression Studio	http://www.microsoft.com/expression/	Commercial and Free (Express)

JavaScript frameworks

Name	Web site
Dojo	http://dojotoolkit.org/
Ext	http://extjs.com/
Spry	http://labs.adobe.com/technologies/spry/
MooTools	http://www.mootools.net/
Prototype	http://www.prototypejs.org/
script.aculo.us	http://script.aculo.us/
QooxDoo	http://qooxdoo.org/
jQuery	http://jquery.com/
MochiKit	http://www.mochikit.com/
Midori	http://www.midorijs.com/
Yahoo UI Toolkit	http://developer.yahoo.com/yui/

WAMP Software

Name	Web site
Apache2Triad	http://apache2triad.net/
EasyPHP	http://www.easyphp.org/
WampServer	http://www.wampserver.com/en/
XAMPP	http://www.apachefriends.org/en/xampp.html

Online learning Resources

Here are some places you can go to learn more on AJAX:

Name	Web site
w3Schools	http://www.w3schools.com/
Lynda.com	http://www.lynda.com/
Adobe developer Connection	http://www.adobe.com/devnet/
Developer Tutorials	http://www.developertutorials.com/
O'Reilly	http://oreilly.com/
Quackit	http://www.quackit.com/

NOTES

NOTES

NOTES

NOTES

NOTES

Corrigendum

The writers for Fast Track to Digital Photography also included Madhusudhan Mukherjee, Rossi Fernandes and Siddharth Parwatay.